Deciding Weak Monadic Second-order Logics using Complex-value Datalog

Gulay Unel and David Toman
{gunel,david}@cs.uwaterloo.ca

Abstract

In this paper we propose to use techniques developed for query evaluation of Complex-value Datalog queries for determining satisfiability of WS1S and WS2S formulæ. This in turn can serve as a decision procedure for Description Logics for which embeddings into WS1S and WS2S have been proposed recently. We show that the use of database query evaluation techniques—in particular the *Magic Set* rewriting of Datalog queries—can considerably improve the performance of automata-based reasoners such as the MONA system.

Keywords: WS1S, WS2S, Automata, Magic Sets, Complex-value Datalog (Datalog^{cv}).

1 Introduction

Recently, WS1S and WS2S reasoners have been employed as reasoners for description logics (in particular, $\mathcal{ALC}$) [15]. The experimental results have shown that MONA—an automata-based WS1S/WS2S reasoner [12, 16]—can be used for testing concept satisfiability. However, for terminological reasoning, the techniques have run into serious state-space explosion problem—the size of a automaton capturing the (language of) models of a given formula quickly exceeds the space available in most computers. This is in stark contrast with (theoretically sub-optimal) tableaux methods that in practice are able to handle much larger problems [11, 13].

This paper introduces an approach that combats this problem. Unlike most other approaches, however, that (usually) attempt to use compression techniques based, e.g., on Binary Decision Diagrams (BDDs) [5] or state space factoring (using, e.g., a *guided* automaton) [16], our approach is based on techniques developed for query evaluation in deductive databases, in particular on the *Magic Set transformation* [2]. In addition we briefly comment on the use of other query optimization techniques such as goal reordering (also known as the join-order selection).

The contributions of the paper are as follows: We show the connection between automata-based decision procedures for WS1S and WS2S and query evaluation in Complex-value Datalog (Datalog^{cv}). Indeed, the complexity of query evaluation in Datalog^{cv} matches the complexity of WS1S and WS2S decision procedures and thus it seems like an appropriate tool for this task. Our approach is based on representing automata using nested relations and on defining the necessary operations on automata as Datalog^{cv} queries. Of particular interest is the fact that the final (non-)emptiness check reduces to posing a (s-t) connectivity query on a nested relational view of the final automaton and can be achieved, e.g., by computing the transitive closure of the representation of the transition relation. This observation allows us to use the Magic Set rewriting to limit the explored state space to elements needed to show non-emptiness.

We have also conducted experiments with CORAL—a Datalog^{cv} system [25]—that show the benefits of the proposed method over more common approaches such as those used by the MONA tool; note that our approach has often outperformed MONA despite the fact that we have used only a naive representation of automata in nested relations. The use of BDDs and other compression techniques seems to be orthogonal to our method and should widen the performance advantage even more[1].

The remainder of the paper is organized as follows. In Section 2, the connection between second order logics and automata is reviewed. Datalog^{cv} and the representation and querying on automata using Datalog^{cv} is outlined in Section 3. In Section 4, the experimental results for the proposed methods are presented. Related work is discussed in Section 5. Finally, conclusions and future research directions are given in Section 6.

2 Logic-Automata Connection

Computational properties of automata provide solutions to many problems. One of these problems is building decision procedures for various logics. In this section we outline the connection between automata and monadic second order logics and focus on constructing automata from formulas. The logic-automaton connection can be generalized to build decision procedures for different logics such as second order logics with one or two successors (S1S or S2S). Automata that accept infinite regular languages can be used for this purpose.

2.1 Formulae of Monadic Second Order Logics

We define the formulas of second order logics as follows.

- The expressions $x = y$, $x = s(y)$, $x \in X$ are atomic formulas, where x,

[1]This, of course, can work only for certain class of problems as we are faced with the non-elementary lower bound in general.

y are individual variables, s is the successor function, and X,Y are set variables.

- Given formulas φ and ϕ, the expressions $\varphi \wedge \phi$, $\varphi \vee \phi$, $\varphi \Rightarrow \phi$, $\neg\varphi$, $\exists x : \varphi$, $\forall x : \varphi$, $\exists X : \varphi$, $\forall X : \varphi$ are also formulas. where x is an individual variable and X is a set variable.
- No other terms are formulas.

The semantics of WS1S is defined on a line; first-order variables are interpreted as natural numbers, and second order variables are interpreted as finite sets of natural numbers. Similarly, the semantics of WS2S is defined over an infinite binary tree $(0+1)^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, ...\}$; first-order variables are interpreted as nodes of the binary tree, and second order variables are interpreted as finite subsets of the nodes. Truth and satisfiability of formulas is defined in the standard way.

2.2 From Formulae to Automata

It is well known that satisfiability of second order logics, e.g., WS1S and WS2S, can be determined using finite automata. The crux of these techniques lies in constructing a finite automaton that accepts exactly the models of a given formula [27]. This technique has been used for showing decidability and for providing tight complexity bounds for many logics. In the case of the above two logics, the automaton is a nondeterministic finite automaton with a finite (or Büchi [6], Rabin [22], etc.) acceptance condition.

In this paper we explore three automata models: finite word automaton, top-down tree automaton, and bottom-up tree automaton. Finite word automaton is the automaton representation of WS1S; and top-down and bottom-up tree automata are models for WS2S. Bottom-up tree automata start their computation at the leaves of the input tree, and top-down tree automata at the root of the input tree in an initial state and then work down the tree level by level simultaneously.

Definition 1 *A finite automaton is a 5-tuple $A = (N_A, X_A, S_A, T_A, F_A)$, where N_A is the set of states (nodes), X_A is the alphabet, S_A is the initial (starting) state, T_A is the transition function, and F_A is the set of final states; where $T_A \subseteq N_A \times X_A \times N_A$ for finite word automata, $T_A \subseteq N_A \times X_A \times N_A \times N_A$ for top-down tree automata, and $T_A \subseteq N_A \times N_A \times X_A \times N_A$ for bottom-up tree automata.*

The automaton is constructed from a given formula inductively. It is well known how to construct automata for the formulas [12, 16]; for example the automaton for the formula $x \in X$ is shown in the left part of Figure 1. Automata representing complex formulas are constructed from simpler ones using automata-theoretic operations.

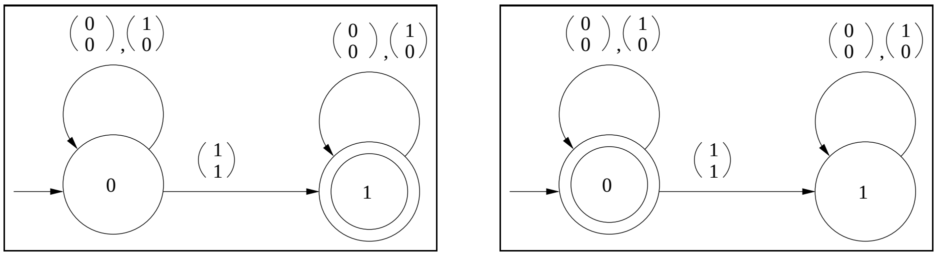

Figure 1: Automata representing the formulae $x \in X$ and $\neg(x \in X)$.

Lemma 1 *Given automata A_φ and A_ϕ representing φ and ϕ, respectively, we can effectively construct an automaton for $\varphi \wedge \phi$, $\varphi \vee \phi$, $\varphi \Rightarrow \phi$, $\neg\varphi$, $\exists x : \varphi$, $\forall x : \varphi$, $\exists X : \varphi$, $\forall X : \varphi$.*

For example, $A_\varphi \oplus A_\phi$, union automaton of A_1 and A_ϕ, accepts $L(A_\varphi) \cup L(A_\phi)$ and represents $\varphi \vee \phi$. The automaton $A_\varphi \times A_\phi$, product automaton of A_φ and A_φ, accepts $L(A_\varphi) \cap L(A_\phi)$ and represents $\varphi \wedge \phi$. The automaton A_φ^c, complement automaton of A_φ, accepts the complement of $L(A_\varphi)$ and represents $\neg\varphi$. Finally, the automaton A_1^p, projection automaton of A_φ, represents $\exists X : \varphi$. Intuitively, the automaton A_φ^p acts as the automaton A_φ for φ except that it is allowed to guess the bits on the track of X.

Example 1 *Let A_1 be an automaton representing the formula $x \in X$. Then the complement automaton A_1^c represents $\neg(x \in X)$ which is shown in the right part of Figure 1.*

We give the actual algorithms for constructing the automata in the following section and in Appendix A (in a Datalog^{cv} syntax).

3 Automata and Datalog for Complex Values

Datalog^{cv} is an extension of Datalog—a language of Horn clauses with variables ranging over constants—with a limited ability to construct terms in the form of tuples and finite sets. To retain termination of query evaluation, the use of terms is restricted in *recursive* clauses [2, 25]. Datalog^{cv} is equivalent to the complex value algebra and calculus in expressive power [1]. Datalog^{cv} programs and queries are defined as follows:

Definition 2 *A* Datalog^{cv} *terms t_i are formed from constants, variables and the tuple ($[t_1, \ldots, t_k]$), set ($\{t_1, \ldots, t_k\}$), and grouping ($< t_1 >$) constructors. A* Datalog^{cv} *atom is a predicate symbol applied to an appropriate number of* Datalog^{cv} *terms. A* Datalog^{cv} *program P is a finite set of Horn clauses of the form $h \leftarrow g_1, \ldots, g_k$, where h (called head) and $g_1, \ldots, g_k$ (called goals) are atoms. The use of the constructors has to be stratified with respect to the program. A* Datalog^{cv} *query is a clause of the form $\leftarrow g_1, \ldots, g_k$. Evaluation of* *a* Datalog^{cv} *query (with respect to* P*) determines whether* $P \models (g_1, \ldots, g_k)\theta$ *for some ground substitution* θ*.*

The evaluation is commonly based on constructing the minimal Herbrand model of P and then determining for which substitutions θ is the query is contained in the model. Furthermore, the extended semi-naive and magic-set techniques can be used to evaluate our queries like the ones proposed in [18] for Relationlog. The main advantage of using Datalog^{cv} is its natural use of fixpoint which allows us to express transitive closure (without the use of an additional power-set operator). Hence, if we transform the automaton and the query representing the satisfiability of a formula to a logic program we can make use of these efficient evaluation techniques.

3.1 Representation of Automata

In this section, we provide a general representation for finite automata.

Definition 3 *The following program* P_A *represents the automaton* $A = (N_A, X_A, S_A, T_A, F_A)$*:*

1. $Node_A(n) \leftarrow$ *(for* $n \in N_A$*),*
2. $Start_A(n) \leftarrow$ *(for* $n \in S_A$*),*
3. $Final_A(n) \leftarrow$ *(for* $n \in F_A$*),*
4. (a) *Automaton for WS1S:*
   $Transition_A(nf_1, nt_1, \overline{x}) \leftarrow$ *(for* $(nf_1, \overline{x}, nt_1) \in T_A$*)*
   (b) *Top-down tree automaton for WS2S:*
   $Transition_A(nf_1, nt_1, nt_2, \overline{x}) \leftarrow$ *(for* $(nf_1, \overline{x}, nt_1, nt_2) \in T_A$*)*
   (c) *Bottom-up tree automaton for WS2S:*
   $Transition_A(nf_1, nf_2, nt_1, \overline{x}) \leftarrow$ *(for* $(nf_1, nf_2, \overline{x}, nt_1) \in T_A$*)*

We have $\overline{x} = \{x_1, x_2, \ldots, x_k\} \subseteq X_A$*, where each* x_i *(for* $1 \leq i \leq k$*) represents an element (a letter) of* X_A*.*

Free variables of the formula represented by A are x_1, x_2, $\ldots$, x_k.

Example 2 *The following program* P_A *represents the automaton* A *shown in the left part of Figure 1:*

$$
\begin{array}{ll}
Node_A(0) \leftarrow & Transition_A(0,\ 0,\ 0,\ 0) \leftarrow \\
Node_A(1) \leftarrow & Transition_A(0,\ 0,\ 1,\ 0) \leftarrow \\
Start_A(0) \leftarrow & Transition_A(0,\ 1,\ 1,\ 1) \leftarrow \\
Final_A(1) \leftarrow & Transition_A(1,\ 1,\ 0,\ 0) \leftarrow \\
 & Transition_A(1,\ 1,\ 1,\ 0) \leftarrow
\end{array}
$$

3.2 Operations on Automata

We define the appropriate automata-theoretic operations: negation, conjunction, projection, and determinization used in decision procedures for the logics under consideration as programs in Datalog^{cv} as follows. Negation is defined in Definition 4, and conjunction, projection, determinization, and transitive closure of the transition function of an automaton are given in Appendix A.

Definition 4 *The following program $P_{\neg A}$ represents the complement automaton A^c of $A = (N_A, X_A, S_A, T_A, F_A)$:*

1. $Node_{\neg A}(n) \leftarrow Node_A(n)$
2. $Start_{\neg A}(n) \leftarrow Start_A(n)$
3. $Final_{\neg A}(n) \leftarrow Node_A(n), \neg Final_A(n)$
4. (a) *Automaton for WS1S:*
   $Transition_{\neg A}(nf_1, nt_1, \overline{x}) \leftarrow Transition_A(nf_1, nt_1, \overline{x})$
   (b) *Top-down tree automaton for WS2S:*
   $Transition_{\neg A}(nf_1, nt_1, nt_2, \overline{x}) \leftarrow Transition_A(nf_1, nt_1, nt_2, \overline{x})$
   (c) *Bottom-up tree automaton for WS2S:*
   $Transition_{\neg A}(nf_1, nf_2, nt_1, \overline{x}) \leftarrow Transition_A(nf_1, nf_2, nt_1, \overline{x})$

Lemma 2 *If A represents α then A^c represents $\neg\alpha$.*

Similar queries can be used to represent the remaining operations on automata including the final (non-)emptiness test. Thus we can construct an automaton A_α corresponding to $\alpha(\theta_1, \theta_2, \ldots, \theta_n)$, where $\theta_1, \theta_2, \ldots, \theta_n$ are the atomic formulas in α, inductively, starting from the atomic formulas and applying the rules given for each operation in Appendix A.

Theorem 1 *Let φ be a WS1S (WS2S) formula. Then φ is satisfiable if and only if $TransClos_{A_\varphi}$ contains a pair consisting of the start and final states.*

Example 3 *Suppose that our formula is $\phi = (\exists Y : Y \subseteq X)$, let A be the automaton for the subformula $Y \subseteq X$, we can use the following logic program to construct the automaton for ϕ:*

$$
\begin{aligned}
&Node_{\exists A}(n) \leftarrow Node_A(n)\\
&Start_{\exists A}(n) \leftarrow Start_A(n)\\
&Final_{\exists A}(n) \leftarrow Final_A(n)\\
&Transition_{\exists A}(n_1, n_2, X) \leftarrow Transition_A(n_1, n_2, X, Y)
\end{aligned}
$$

The above clauses define a non-deterministic automaton ($\exists A$) representing the formula (see Definition 6).

$Node_{D_{\exists A}}(\{\}) \leftarrow$
$SNode_{D_{\exists A}}(\{n\}) \leftarrow Node_{\exists A}(n)$
$Node_{D_{\exists A}}(N) \leftarrow SNode_{D_{\exists A}}(N_1), Node_{D_{\exists A}}(N_2), Union(N_1, N_2, N)$
$Start_{D_{\exists A}}(\{n\}) \leftarrow Start_{\exists A}(n)$
$Final_{D_{\exists A}}(N) \leftarrow Node_{D_{\exists A}}(N), Final_{\exists A}(n), member(n, N)$
$Transition_{D_{\exists A}}(N_1, < n_2 >, X) \leftarrow Transition_{\exists A}(n_1, n_2, X),$
$\qquad Node_{D_{\exists A}}(N_1), member(n_1, N_1)$
$TranClos_{D_{\exists A}}(n_1, n_2) \leftarrow Transition_{D_{\exists A}}(n_1, n_2, X)$
$TranClos_{D_{\exists A}}(n_1, n_2) \leftarrow Transition_{D_{\exists A}}(n_1, n_3, X), TranClos_{D_{\exists A}}(n_3, n_2)$

The remaining clauses convert the automaton to a deterministic automaton ($D_{\exists A}$) representing the formula (see Definition 7), and compute the transitive closure of its transition function (see Definition 8) in order to test for non emptiness. The final query thus is:

$$\leftarrow Start_{D_{\exists A}}(N), Final_{D_{\exists A}}(M), TranClos_{D_{\exists A}}(N, M).$$

Magic-set rewriting [3, 20]—a well known query optimization method—is applied to the above program prior to query evaluation. The Magic-set technique improves the bottom-up evaluation such that its performance rivals the efficiency of the top-down evaluation. The idea behind the magic-set technique lies in restricting the computation of intermediate results to those facts that are needed to answer a query.

The left part of Figure 2 illustrates the bottom-up evaluation of the program given in Example 3, the right part illustrates the magic-set evaluation of the same program. The effect of the Magic set-based query evaluation is even more

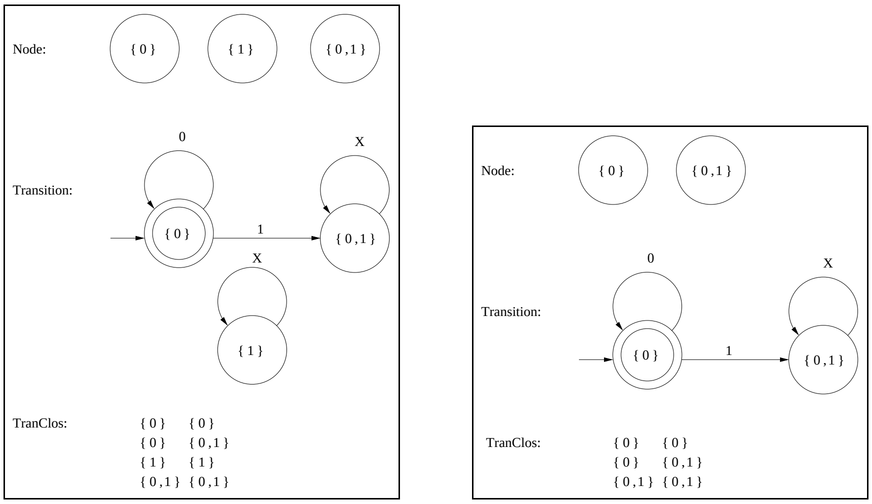

Figure 2: Bottom-up and Magic Set evaluation of the program in Example 3.

pronounced for larger formulas. For example, for the formula

$$\phi = (x \in X) \wedge (\exists Y : (y \in Y) \wedge (z \in Y))$$

the bottom-up evaluation creates 1536 nodes, 24528 transitions while the magic-set evaluation technique creates only 2 nodes and 1 transition.

4 Experiments

We compare our technique with the MONA system [12, 16], one of the most efficient tools for reasoning of weak second order logics (WS1S and WS2S). In contrast to MONA which constructs the whole automaton for a given formula our method only constructs the nodes we need to answer the non emptiness query. For the experiments we use CORAL, a deductive system that supports Datalog^{cv} and Magic sets.

The performance results are given in Figure 3. The response times are measured in seconds; N/A means "Not Answered". The formulas are similar to the ones in T98 satisfiability test suite except we varied their sizes, in particular the number of existential quantifiers and free variables. We have observed that the magic-set evaluation method exhibits a considerable performance gain over MONA for formulas with many free variables. On the other hand, MONA usually performs better than CORAL for the formulas with many existential quantifiers. We believe that this is a problem caused by the implementation CORAL uses for the evaluation of programs with sets (and can be avoided using a more sophisticated implementation of Datalog^{cv}). In addition MONA uses the BDD data structures and algorithms to enhance its performance. Our current implementation in CORAL does not support these structures and is likely to perform better using them.

	1	2	3	4	5	6	7	8	9	10
MONA	N/A	N/A	7.08	7.14	5.22	4.37	13.33	N/A	10.80	N/A
CORAL	4.15	5.22	4.76	11.94	3.95	2.42	4.75	16.25	3.78	12.09

Figure 3: Performance results (secs)

We also began exploring the impact of goal reordering on the performance of the Datalog^{cv} program representations of the automata. The following example illustrates the importance of technique:

Example 4 *We show three rewritings of a formula and their performance results. Consider the following formulas:*

$$\begin{aligned}
\varphi_1 &= (x_{11} \in Y_{11}) \iff (x_{12} \in Y_{12}) \\
\varphi_2 &= (\exists Y_{13} : (x_{13} \in Y_{13}) \wedge (x_{14} \in Y_{13}) \\
\varphi_3 &= (((x_1 \in Y_1) \iff (x_2 \in Y_2)) \Rightarrow ((x_3 \in Y_3) \iff (x_4 \in Y_4))) \wedge \\
&\quad (((x_5 \in Y_5) \iff (x_6 \in Y_6)) \Rightarrow ((x_7 \in Y_7) \iff (x_8 \in Y_8))) \wedge \\
&\quad ((x_9 \in Y_9) \iff (x_{10} \in Y_{10}))
\end{aligned}$$

Then the goal ordering results in the following timing which shows that the minimal response time we get using CORAL is 16.34 seconds, MONA fails in all three cases.

	$\varphi_1 \wedge \varphi_2 \wedge \varphi_3$	$\varphi_3 \wedge \varphi_1 \wedge \varphi_2$	$\varphi_3 \wedge \varphi_2 \wedge \varphi_1$
MONA	N/A	N/A	N/A
CORAL	N/A	16.34	31.19

Figure 4: Performance results (secs)

5 Related Work

The connection between logic and automata was first considered by Büchi [6] and Elgot [10]. They have shown that monadic second-order logic over finite words and finite automata have the same expressive power, and we can transform formulas of this logic to finite automata and vice versa. Later, Büchi [7], McNaughton [19], and Rabin [22] proved that monadic second-order logic over infinite words (and trees) and finite automata also have the same expressive power. The practical use of this connection was investigated for temporal logics and fixed-point logics which led to the theory of model checking [4, 29]. Another automata theoretic construction used in specification and verification was for μ-calculus [14] and description logics [30]. An extensive survey on automata and logic can be found in [27].

The logic-automaton connection has been used for implementing decision procedures for various logics. It is argued that the success of these procedures relies on efficient operations on a compact representation of automata based on BDDs [16, 17].

We have used deductive techniques to represent and query automata. The system used to support our implementation, CORAL [23, 24, 25], provides efficient set-oriented data manipulation common in relational systems. There are numerous other deductive systems which support logic-programming languages with sets and tuples, e.g., LDL [9, 21], and XSB [26] (here sets have to be explicitly simulated). In terms of the evaluation strategy XSB uses top-down evaluation with memoing whereas CORAL uses magic sets.

Considerable work has been done on query optimization in relational and deductive database systems [20]. Query optimization in relational systems includes choosing join orders and cost models [8, 28]. We use the idea of magic sets rewriting [3] as a deductive database optimization method for our queries, and we are planning to use cost-based optimization methods to improve our query evaluation.

6 Conclusions and Future Work

In this paper we have presented a translation technique that maps satisfiability questions for formulas in WS1S and WS2S and, in turn, implication problems in $\mathcal{ALC}$ to query evaluation in Datalog^{cv}. The connection was made using the link between logic and finite automata. We have shown how the evaluation techniques used for answering queries over these programs provide efficient decision procedures for second order logics.

Future extensions of the proposed approach include extending the translation to other types of automata on infinite objects, e.g., to Rabin [22] and Alternating Automata [30], and on improving the upper complexity bounds by restricting the form of Datalog^{cv} programs generated by the translation (when used for decision problems in, e.g., EXPTIME). In all these cases, the goal is to match the optimal theoretical bounds while avoiding the worst-case behavior (inherent in most automata-based techniques) in as many situations as possible. In addition we plan to study the impact of goal reordering and various other query optimization techniques on the performance of the decision procedure and to develop heuristics (patterned on cost-based join-order and query optimization) for this purpose. We also plan to compare the CORAL-based implementations with implementations based on the XSB system, a logic programming system with memoing [26].

References

[1] S. Abiteoul, C. Beeri, "The Power of Languages for the Manipulation of Complex Values", *VLDB Journal*, Vol. 4, No. 4, pp. 727-794, 1995.

[2] C. Beeri, S. Naqvi, O. Shmueli, S. Tsur, "Set Construction in a Logic Database Language", *Journal of Logic Programming*, Vol. 10, No. 3&4, pp. 181-232, 1991.

[3] C. Beeri, R. Ramakrishnan, "On the Power of Magic", *Journal of Logic Programming*, Vol. 10, Nos. 1–4, pp. 255-299, 1991.

[4] O. Bernholtz, M. Y. Vardi, P. Wolper, "An Automata-theoretic Approach to Branching-time Model Checking", *Computer Aided Verification, Proc. 6th Int. Workshop*, pp. 142-155, 1994.

[5] R. E. Bryant, "Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams", *ACM Computing Surveys*, Vol. 24, No. 3, pp. 293-318, 1992.

[6] J. R. Büchi, "Weak Second-order Arithmetic and Finite Automata", *Z. Math. Logik Grundl. Math.*, Vol. 6, pp. 66-92, 1960.

[7] J. R. Büchi, "On a Decision Method in Restricted Second-order Arithmetic", *Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science*, pp. 1-11, 1962.

[8] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems", *PODS*, pp. 34-43, 1998.

[9] D. Chimenti, R. Gamboa, R. Krishnamurthy, S. A. Naqvi, S. Tsur, C. Zaniolo, "The LDL System Prototype", *IEEE Trans. Knowl. Data Eng.*, Vol. 2, No. 1, pp. 76-90, 1990.

[10] C. C. Elgot, "Decision Problems of Finite Automata Design and Related Arithmetics", *Trans. Amer. Math. Soc.*, Vol. 98, pp. 21-52, 1961.

[11] V. Haarslev, R. Möller, "High performance reasoning with very large knowledge bases: A practical case study", *IJCAI*, pp. 161-166, 2001.

[12] J. G. Henriksen, J. L. Jensen, M. E. Jörgensen, N. Klarlund, R. Paige, T. Rauhe, A. Sandholm, "Mona: Monadic Second-Order Logic in Practice", *TACAS*, pp. 89-110, 1995.

[13] I. Horrocks, "Using an expressive description logic: FaCT or fiction?", *Knowledge Representation and Reasoning (KR)*, pp. 636-647, 1998.

[14] D. Janin, I. Walukiewicz, "Automata for the Modal μ-Calculus and related Results", *MFCS*, pp. 552-562, 1995.

[15] E. Karabaev, C. Lutz, "Mona as a DL Reasoner", *Description Logics*, 2004.

[16] N. Klarlund, "Mona & Fido: The Logic-Automaton Connection in Practice", *Computer Science Logic*, pp. 311-326, 1997.

[17] N. Klarlund, A. Møller, M. I. Schwartzbach, "MONA Implementation Secrets", *Int. J. Found. Comput. Sci.*, Vol. 13, No. 4, pp. 571-586, 2002.

[18] M. Liu, "Query Processing in Relationlog", *DEXA*, pp. 342-351, 1999.

[19] R. McNaughton, "Testing and Generating Infinite Sequences by a Finite Automaton", *Information and Control*, Vol. 9, pp. 521-530, 1966.

[20] I. S. Mumick, "Query Optimization in Deductive and Relational Databases", *PhD Thesis*, *Department of Computer Science, Stanford University*, 1991.

[21] S. Naqvi, S. Tsur, "A Logical Language for Data and Knowledge Bases", *Computer Science Press*, 1989.

[22] M. O. Rabin, "Decidability of Second-order Theories and Automata on Infinite Trees", *Trans. Amer. Math. Soc.*, Vol. 141, pp. 1-35, 1969.

[23] R. Ramakrishnan, P. Bothner, D. Srivastava, S. Sudarshan, "CORAL–A Database Programming Language", *Workshop on Deductive Databases*, 1990.

[24] R. Ramakrishnan, D. Srivastava, S. Sudarshan, "CORAL–Control, Relations and Logic", *VLDB Journal*, pp. 238-250, 1992.

[25] R. Ramakrishnan, D. Srivastava, S. Sudarshan, P. Seshadri, "The CORAL Deductive System", *VLDB Journal*, Vol. 3, No. 2, pp. 161-210, 1994.

[26] K. F. Sagonas, T. Swift, D. S. Warren, "XSB as an Efficient Deductive Database Engine", *SIGMOD Conference*, pp. 442-453, 1994.

[27] W. Thomas, "Languages, Automata, and Logic", *Handbook of Formal Languages*, Vol. 3, 1997.

[28] J. D. Ullman, "Principles of Database and Knowledge-Base Systems", *Computer Science Press*, Vol. 1&2, 1989.

[29] M. Y. Vardi, P. Wolper, "An Automata-theoretic Approach to Automatic Program Verification", *Proc. of the First Symposium on Logic in Computer Science*, pp. 322-331, 1986.

[30] M. Y. Vardi, "Reasoning about The Past with Two-Way Automata", *ICALP*, pp. 628-641, 1998.

A Appendix

In this section, we define conjunction, projection, determinization, and transitive closure of the transition function of an automaton. Conjuction operation is provided in Definition 5, projection in Definition 6, determinization in Definition 7, and finally transitive clossure of the transition function is given in Definition 8.

Definition 5 *The following program $P_{A_1 \wedge A_2}$ represents the product automaton $A_1 \times A_2$ of $A_1 = (N_{A_1}, X_{A_1}, S_{A_1}, T_{A_1}, F_{A_1})$, and $A_2 = (N_{A_2}, X_{A_2}, S_{A_2}, T_{A_2}, F_{A_2})$:*

1. $Node_{A_1 \wedge A_2}([n_1, n_2]) \leftarrow Node_{A_1}(n_1), Node_{A_2}(n_2)$
2. $Start_{A_1 \wedge A_2}([n_1, n_2]) \leftarrow Start_{A_1}(n_1), Start_{A_2}(n_2)$
3. $Final_{A_1 \wedge A_2}([n_1, n_2]) \leftarrow Final_{A_1}(n_1), Final_{A_2}(n_2)$
4. (a) *Automaton for WS1S:*
   $Transition_{A_1 \wedge A_2}([nf_{11}, nf_{21}], [nt_{11}, nt_{21}], \overline{x}, \overline{y}, \overline{z}) \leftarrow$
   $\quad Transition_{A_1}(nf_{11}, nt_{11}, \overline{x}, \overline{y}), Transition_{A_2}(nf_{21}, nt_{21}, \overline{y}, \overline{z})$

   (b) *Top-down tree automaton for WS2S:*
   $Transition_{A_1 \wedge A_2}([nf_{11}, nf_{21}], [nt_{11}, nt_{21}], [nt_{12}, nt_{22}], \overline{x}, \overline{y}, \overline{z}) \leftarrow$
   $\quad Transition_{A_1}(nf_{11}, nt_{11}, nt_{12}, \overline{x}, \overline{y}),$
   $\quad Transition_{A_2}(nf_{21}, nt_{21}, nt_{22}, \overline{y}, \overline{z})$

   (c) *Bottom-up tree automaton for WS2S:*
   $Transition_{A_1 \wedge A_2}([nf_{11}, nf_{21}], [nf_{12}, nf_{22}], [nt_{11}, nt_{21}], \overline{x}, \overline{y}, \overline{z}) \leftarrow$
   $\quad Transition_{A_1}(nf_{11}, nf_{12}, nt_{11}, \overline{x}, \overline{y}),$
   $\quad Transition_{A_2}(nf_{21}, nf_{22}, nt_{21}, \overline{y}, \overline{z})$

Here, $\overline{x}$, $\overline{y}$ represent the free variables of the formula A_1 represents, and $\overline{y}, \overline{z}$ of the formula A_2 represents.

Lemma 3 *If A_1 represents α_1, and A_2 represents α_2 then, $A_1 \times A_2$ represents $\alpha_1 \wedge \alpha_2$.*

Definition 6 *The following program $P_{\exists A}$ represents the projection automaton A^p of $A = (N_A, X_A, S_A, T_A, F_A)$:*

1. $Node_{\exists A}(n) \leftarrow Node_A(n)$
2. $Start_{\exists A}(n) \leftarrow Start_A(n)$
3. $Final_{\exists A}(n) \leftarrow Final_A(n)$
4. (a) *Automaton for WS1S:*
   $Transition_{\exists A}(nf_1, nt_1, \overline{y}) \leftarrow Transition_A(nf_1, nt_1, \overline{x}, \overline{y})$

   (b) *Top-down tree automaton for WS2S:*
   $Transition_{\exists A}(nf_1, nt_1, nt_2, \overline{y}) \leftarrow Transition_A(nf_1, nt_1, nt_2, \overline{x}, \overline{y})$

   (c) *Bottom-up tree automaton for WS2S:*
   $Transition_{\exists A}(nf_1, nf_2, nt_1, \overline{y}) \leftarrow Transition_A(nf_1, nf_2, nt_1, \overline{x}, \overline{y})$
5. (a) *Automaton for WS1S:*

   i. $Transition_X(nf_1, nt_1, \overline{x}, \overline{o}) \leftarrow Transition_A(nf_1, nt_1, \overline{x}, \overline{y})$
   ii. $Final_{\exists A}(n) \leftarrow Node_A(n), Transition_X(nf_1, nt_1, \overline{x}, \overline{y}), Final_A(nt_1),$
   $\quad Transition_A(n, nt_2, \overline{x}, \overline{y}), Final_A(nt_2)$

(b) *Top-down tree automaton for WS2S:*

   i. $Transition_X(nf_1, nt_1, nt_2, \overline{x}, \overline{o}) \leftarrow Transition_A(nf_1, nt_1, nt_2, \overline{x}, \overline{y})$

   ii. $Final_{\exists A}(n) \leftarrow Node_A(n),$
   $Transition_X(nf_1, nt_1, nt_2, \overline{x}, \overline{y}),\ Final_A(nt_1),\ Final_A(nt_2),$
   $Transition_A(n, nt_3, nt_4, \overline{x}, \overline{y}),\ Final_A(nt_3),\ Final_A(nt_4)$

(c) *Bottom-up tree automaton for WS2S:*

   i. $Transition_X(nf_1, nf_2, nt_1, \overline{x}, \overline{o}) \leftarrow Transition_A(nf_1, nf_2, nt_1, \overline{x}, \overline{y})$

   ii. $Final_{\exists A}(n) \leftarrow Node_A(n),\ Transition_X(nf_1, nf_2, nt_1, \overline{x}, \overline{y}),$
   $Final_A(nt_1),\ Transition_A(n, nf_3, nt_2, \overline{x}, \overline{y}),\ Final_A(nt_2)$

   iii. $Final_{\exists A}(n) \leftarrow Node_A(n),\ Transition_X(nf_1, nf_2, nt_1, \overline{x}, \overline{y}),$
   $Final_A(nt_1),\ Transition_A(nf_3, n, nt_2, \overline{x}, \overline{y}),\ Final_A(nt_2)$

Here, $\overline{o} = \{0, 0, \ldots, 0\}$ where $|\overline{o}| = |\overline{y}|$, $\overline{y}$ represents free variables, and $\overline{x}$ represents bound variables of the formula represented by A.

Lemma 4 *If A represents α then A^p represents $\exists x_1, x_2, \ldots, x_k : \alpha$ where A^p is the projection automaton of A.*

Definition 7 *The following program P_{D_A} represents the determinized automaton A^d of $A = (N_A,\ X_A,\ S_A,\ T_A,\ F_A)$:*

1. (a) $Node_{D_A}(\{\}) \leftarrow$
   (b) $SNode_{D_A}(\{n\}) \leftarrow Node_A(n)$
   (c) $Node_{D_A}(N) \leftarrow SNode_{D_A}(N_1),\ Node_{D_A}(N_2),\ Union(N_1, N_2, N)$
2. $Start_{D_A}(\{n\}) \leftarrow Start_A(n)$
3. $Final_{D_A}(N) \leftarrow Node_{D_A}(N),\ Final_A(n),\ member(n, N)$
4. (a) *Automaton for WS1S:*
   $Transition_{D_A}(Nf_1, < nt_1 >, \overline{x}) \leftarrow Transition_A(nf_1, nt_1, \overline{x}),$
   $Node_{D_A}(Nf_1),\ member(nf_1, Nf_1)$
   (b) *Top-down tree automaton for WS2S:*
   $Transition_{D_A}(Nf_1, < nt_1 >, < nt_2 >, \overline{x}) \leftarrow Transition_A(nf_1, nt_1, nt_2, \overline{x}),$
   $Node_{D_A}(Nf_1),\ member(nf_1, Nf_1)$
   (c) *Bottom-up tree automaton for WS2S:*
   $Transition_{D_A}(Nf_1, Nf_2, < nt_1 >, \overline{x}) \leftarrow Transition_A(nf_1, nf_2, nt_1, \overline{x}),$
   $Node_{D_A}(Nf_1),\ member(nf_1, Nf_1),$
   $Node_{D_A}(Nf_2), member(nf_2, Nf_2)$

The function $Union(N_1, N_2, N)$ takes two sets N_1 and N_2 as inputs and assigns their union to N, $member(n, N)$ checks if n is a member of the set N or not.

Lemma 5 *If A represents α then A^d represents α, and A^d is the determinized A.*

Definition 8 *The following program P_{TC_A} computes the transitive closure of the transition function of A to find out if the language A represents is non-empty or not, and as a result, if α, which is the formula represented by A, is satisfiable or not.*

1. *Automaton for WS1S:*

   (a) $TranClos_A(nf_1, nt_1) \leftarrow Transition_A(nf_1, nt_1, \overline{x})$
   (b) $TranClos_A(nf_1, nt_1) \leftarrow$
   $Transition_A(nf_1, nt_2, \overline{x}),\ TranClos_A(nt_2, nt_1)$

2. *Top-down tree automaton for WS2S:*

   (a) $TranClos_A(nf_1, nt_1, nt_2) \leftarrow Transition_A(nf_1, nt_1, nt_2, \overline{x})$
   (b) $TranClos_A(nf_1, nt_1, nt_2) \leftarrow$
   $Transition_A(nf_1, nt_3, nt_4, \overline{x}),\ TranClos_A(nt_3, nt_1, nt_2)$
   (c) $TranClos_A(nf_1, nt_1, nt_2) \leftarrow$
   $Transition_A(nf_1, nt_3, nt_4, \overline{x}),\ TranClos_A(nt_4, nt_1, nt_2)$

3. *Bottom-up tree automaton for WS2S:*

   (a) $TranClos_A(nf_1, nf_2, nt_1) \leftarrow$
   $Transition_A(nf_1, nf_2, nt_1, \overline{x})$
   (b) $TranClos_A(nf_1, nf_2, nt_1) \leftarrow$
   $Transition_A(nf_1, nf_2, nt_2, \overline{x}),\ TranClos_A(nt_2, nf_3, nt_1)$
   (c) $TranClos_A(nf_1, nf_2, nt_1) \leftarrow Transition_A(nf_1, nf_2, nt_2, \overline{x}),$
   $TranClos_A(nf_3, nt_2, nt_1)$