Datalog Trees and Their Automata

Foto N. Afrati **Eugenie Foustoucos***
Theodore Andronikos
National Technical University of Athens
afrati@cs.ece.ntua.gr eugenie@ermis.cs.ece.ntua.gr tedandr@dsclab.ntua.gr

1 Introduction

Much attention has been given the past fifteen years to Datalog, the well-known database query language which belongs to the family of logic programming languages with well-defined operational semantics [GMN84,Ull89,AHV95,Min96]. These semantics are described either in terms of fixpoint operators or in terms of the resolution inference rule [Rob65,Llo87]. Syntactically Datalog can be seen as Prolog without function symbols and this absence of function symbols allows to define operational semantics in a simpler way, since for instance the full power of unification is not needed. It is not easy however to understand in what sense things become simpler for Datalog, and one reason for that is the variety of trees formalisms used in the proof-theoretic approach to Datalog: SLD-trees, expansion trees, unfolding proof trees, proof trees, pseudo-proof trees, trace trees, skeleton trees are some of the trees we encouter in the field [Ull85,Ull89,AHV95,Af97,Var92,Var94]. All of them express more or less the resolution-based operational semantics of Datalog programs. Manipulation of variables is a central point in the definition of operational semantics and it is also a delicate operation; the difference between the trees mentioned above lies often in the way this manipulation is performed.

In the present paper we define a new tree called Datalog tree expressing top-down query evaluation in a variable-free manner which appears to offer many advantages. For an early version of Datalog trees under the name of legitimate trees, see [FLS95,Fou96]. We also define tree automata recognizing our Datalog trees. The use of automata theory in the context of Datalog has started to be investigated rather recently. The description of Datalog trees in terms of tree automata becomes formally correct after non-trivial subtleties are taken into account.

The theory of finite automata is well-established and well-known. Since its introduction in the late 1950's, the theory of finite automata, now well-established and well-known, has found many important applications in most areas of computer science (e.g., compiler construction). Finite automata are traditionally viewed either as formal models of computation or acceptors of regular languages. However, finite automata can be used to define more complex objects, such as trees and infinite words (see [Tho96] for a recent survey). In this respect, we shall use tree automata as succinct devices that recognize Datalog trees. It should come as no surprise, considering the work of Vardi ([Var92,Var94]) in this area, that tree automata have adequate expressive power to capture the desired properties of Datalog programs.

*Research supported by a Fellowship of the Greek Scholarship Foundation (IKY).

2 Pattern transformations

Definition 2.1 We call *pattern transformation* any total function from $\{1, ..., q\}$ to $\{1, ..., p, \star\}$, where $\star$ is a special symbol and p, q are positive integers. The set $\{1, ..., q\}$ is the domain of ϕ and $\{1, ..., p, \star\}$ is the image of ϕ.

Any pattern transformation ϕ with graph $\{(1, \phi(1)), ..., (q, \phi(q))\}$ will be written in a "substitution-like" manner as $\phi = \{1|\phi(1), ..., q|\phi(q)\}$ [1]. If $\phi(i) = i$ for all $i = 1, ..., q$ then ϕ is the identity function. A pattern transformation ϕ need not be an onto function. If $\phi = \{1|\phi(1), ..., q|\phi(q)\}$ and $max\{\phi(1), .., \phi(q)\} = p$ then ϕ can be considered of (having) image $\{1, ..., p\}$ and also of image $\{1, ..., p'\}$ for any $p' > p$. According to what is considered to be the image of ϕ we say that ϕ is of type (p, q) or (p', q) $(p' > p)$.

Proposition 2.2 Every pair (X, Y) of variable vectors where $X = (x_1, ..., x_p)$ and $Y = (y_1, ..., y_q)$ defines a pattern transformation θ of type (p, q) as follows:
$\theta(i) = j$ if j is the least index such that $y_i = x_j$ and $\theta(i) = \star$ if $y_i \neq x_j$ $\forall j = 1, ..., p$.

Example 2.3 The pair $((x, y), (y, u, x, x))$ defines the pattern transformation $\theta = \{1|2, 2|\star, 3|1, 4|1\}$ of type $(2, 4)$. θ is also defined by the pair $((x, y, z), (y, u, x, x))$ and then it is of type $(3, 4)$.
The pair $((x, y), (y, v))$ defines the pattern transformation $\phi = \{1|2, 2|\star\}$ of type $(2, 2)$. ϕ is also defined by the pair $((x, y, z), (y, v))$ and it is then of type $(3, 2)$.

Definition 2.4 Let V be an countable set of variables.
Every pattern transformation θ of type (p, q) defines a function $\overline{\theta}$ from V^p to $\mathcal{P}(V^q)$ such that for any $(x_1, ..., x_p) \in V^p$, $\overline{\theta}((x_1, ..., x_p)) = \{(x_{\theta(1)}, ..., x_{\theta(q)})|x_\star \in V \setminus \{x_1, ..., x_p\}\}$.

Example 2.5 Let $\theta = \{1|2, 2|\star, 3|1, 4|1\}$ and $V = \{x, y, z, u, v, w, ...\}$.
We underline the variables obtained from the symbol $\star$.
$\overline{\theta}(x, y, z) = \{(y, \underline{u}, x, x), (y, \underline{v}, x, x), (y, \underline{w}, x, x), ...\} = \overline{\theta}(x, y, x) = \overline{\theta}(x, y, y)$.
$\overline{\theta}(x, x, z) = \{(x, \underline{u}, x, x), (x, \underline{v}, x, x), ...\}$ and $\overline{\theta}(u, v, w) = \{(v, \underline{x}, u, u), (v, \underline{y}, u, u), ...\}$.

Since all q-tuples in $\overline{\theta}(X)$ are equal up to renaming, we shall assimilate $\overline{\theta}(X)$ with any one of its elements. Notice then that all elements of the graph $\mathcal{G}$ of $\overline{\theta}$ define the same pattern transformation θ.

3 Datalog trees

A *Datalog program* Π is a collection of rules of the form $h :- g_1, \ldots, g_\kappa$ where h is the *head*, and $g_1, \ldots, g_\kappa$ form the *body* of the rule. Each g_i is an expression of the form $R(x_1, \ldots, x_m)$, where the x_i's are variables and R is either an *extensional database* (EDB) predicate naming one of the

[1] Note that generally $\phi = \{1|\phi(1), ..., q|\phi(q)\}$ is not a substitution (since it can happen that for some $i = 1, ..., q$ $\phi(i) = i$). We trivially obtain a substitution by restricting the domain of definition $\{1, ..., q\}$ of ϕ to those elements j such that $\phi(j) \neq j$.

database relations, or an *intensional database* (IDB) predicate which is defined by the program; IDB predicates are exactly the ones appearing in the heads of rules. If R is an EDB (IDB) predicate, then $R(x_1, \ldots, x_m)$ is an EDB (IDB)-*atom*. We usually denote by EDB_1,...,EDB_s the EDB-atoms occurring in the body of a rule. We call *initialization* rule a rule with no IDB predicate in its body; any non initialization rule is called *recursive* rule. We adopt the following convention about a program Π with K rules: r_1,...,r_k are the k recursive rules of Π and r_{k+1},...,r_K are the $K-k$ initialization rules of Π. In the present work we consider, for technical reasons, that Datalog programs are in the following normal form.

Definition 3.1 A Datalog program Π is *normal* if the following conditions are satisfied for every rule r of Π:
1. every variable occurring in at least two IDB atoms of the body of r must appear in the head of r and
2. every variable with at least two occurrences in some IDB atom of the body of r must appear in the head of r.

Any Datalog program Π can indeed be transformed in a normal one by making the following two simple changes for every rule r (with head $P(x_1, ..., x_n)$) of Π violating conditions 1 and/or 2 above: a) transform r into a new rule r' by replacing the head of r with $P'(x_1, ..., x_n, u_1, ..., u_p)$ such that conditions 1 and 2 are satisfied (P' is a new predicate symbol and u_1,...,u_p are the variables responsible for the violation of conditions 1, 2 and b) add to Π the new rule $P(x_1, ..., x_n) \leftarrow P'(x_1, ..., x_n, u_1, ..., u_p)$. It is not hard to prove that every Datalog program is equivalent to a normal one produced by the above transformation. In the rest of the paper we will deal with normal Datalog programs. For the rigorous definition of the operational semantics of Datalog programs we refer to well-known text books [CGT90,AHV95,Ull89].

Definition 3.2 Let Π be a Datalog program with k recursive rules r_1,...,r_k and for $i = 1, .., k$ let i_{max} be the number of IDB atoms in r_i. The *transformation set* $\mathcal{S}_\Pi$ of Π is the set $\{\sigma_{(i,j)}, i = 1, ..., k;\ j = 1, ..., i_{max}\}$ where $\sigma_{(i,j)}$ denotes a specific pattern transformation related to the variable list of the j^{th} IDB atom of the body of r_i. The pattern transformation denoted by $\sigma_{(i,j)}$ is defined by the pair $((x_1, ..., x_p), (y_1, ..., y_q))$ where $P_i(x_1, ..., x_p)$ is the head of r_i and $Q_j(y_1, ..., y_q))$ is the j^{th} IDB atom in the body of r_i.

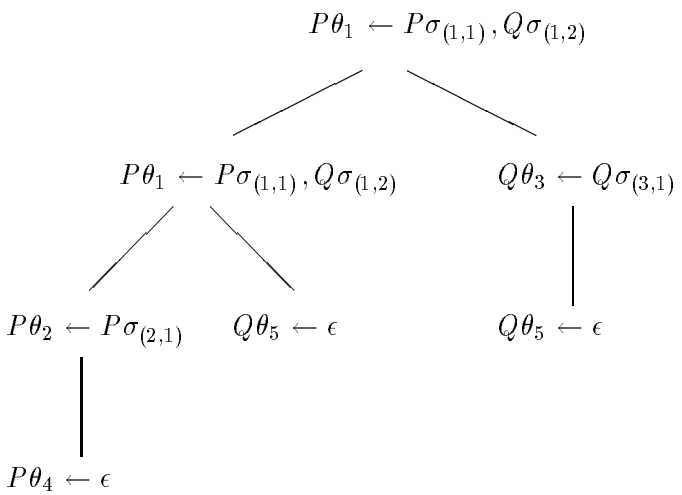
Definition 3.3 Let Π be a Datalog program with K rules r_1,...,r_K. The *constraint set* $\mathcal{R}_\Pi$ of Π is the set of pairs (P_i, θ_i) for $i = 1, ..., K$ where P_i is the head predicate of the i^{th} rule r_i and θ_i denotes a specific pattern transformation related to the variable list of the head of r_i. The pattern transformation denoted by θ_i is computed as follows: if r_i has head $P_i(x_1, ..., x_p)$ where $(x_1, ..., x_p)$ is a vector of distinct variables, then θ_i is the identity function; if r_i has head $P_i(x_1, ..., x_p)$ where $(x_1, ..., x_p)$ is a vector of variables, not all pairwise distinct, then θ_i is the pattern transformation defined by any pair $((z_1, z_2, ..., z_p), (x_1, ..., x_p))$ such that for all i, the variables z_i are distinct and $z_i = x_i$ if for all $j < i$ $x_j \neq x_i$.

Definition 3.4 Let Π be a Datalog program. To each rule r_i of Π we associate the *abstracted* rule R_i defined as follows: if $r_i : P(\vec{x}) \leftarrow P_1(\vec{x}_1), ..., P_n(\vec{x}_n), EDB_1, ..., EDB_s$ is a recursive rule then $R_i : P\theta_i \leftarrow P_1\sigma_{(i,1)}, ..., P_n\sigma_{(i,n)}$ and if $r_i : P(\vec{x}) \leftarrow EDB_1, ..., EDB_s$ is an initialization rule then $R_i : P\theta_i \leftarrow \epsilon$ where ϵ is a new symbol. P is the head predicate of r_i and of R_i, P_j

for $j = 1, ..., n$ is the predicate of the j^{th} IDB atom of r_i and of R_i, θ_i is defined according to definition 3.3 and $\sigma_{(i,j)}$ is defined according to definition 3.2. Thus a Datalog program with K rules r_1,...,r_K has K abstracted rules R_1,...,R_K.

Definition 3.5 A Datalog-Q-tree of a Datalog program Π with K rules and k recursive rules is a tree with nodes labeled by abstracted rules of Π such that:
- the root label is an abstracted rule with head predicate the goal predicate Q,
- a node N with label $P\theta_i \leftarrow P_1\sigma_{(i,1)}, ..., P_n\sigma_{(i,n)}$ has n sons N_1,...,N_n with respective labels an abstracted rule with head predicate P_j, $j = 1, ..., n$.
- a node N with label $P\theta_i \leftarrow \epsilon$ is a leaf.

Example 3.6 Let Π be a Datalog program consisting of three recursive rules r_1, r_2, r_3 and two initialization rules r_4 and r_5:
$r_1 : P(x, y, z) \leftarrow P(y, z, w), Q(x, z), E(y, w)$
$r_2 : P(x, x, z) \leftarrow P(x, u, z), E(u, x)$
$r_3 : Q(x, y) \leftarrow E(x, z), Q(z, y)$
$r_4 : P(x, y, z) \leftarrow A(x, y, z)$
$r_5 : Q(x, y) \leftarrow E(x, y)$.
Let $\mathcal{S}_\Pi = \{\sigma_{(1,1)}, \sigma_{(1,2)}, \sigma_{(2,1)}, \sigma_{(3,1)}\}$ be the transformation set of Π where $\sigma_{(1,1)} = \{1|2, 2|3, 3|\star\}$, $\sigma_{(1,2)} = \{1|1, 2|3\}$, $\sigma_{(2,1)} = \{1|1, 2|\star, 3|3\}$ and $\sigma_{(3,1)} = \{1|\star, 2|2\}$. The pattern transformation $\sigma_{(1,2)}$ is related to the variable list (x, z) of the second IDB atom of the body of r_1 and is defined by the pair $((x, y, z), (x, z))$.
Let $\mathcal{R}_\Pi = \{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$ be the constraint set of Π where $\theta_1 = \theta_3 = \theta_4 = \theta_5 = id$ (id is the identity function) and $\theta_2 = \{1|1, 2|1, 3|3\}$. The pattern transformation θ_2 is related to the variable list (x, x, z) of the head of r_2 and it is defined by the pair $((x, \underline{y}, z), (x, x, z))$ (or by the pair $((x, \underline{w}, z), (x, x, z))$). A Datalog-$P$-tree of program Π is designed in the figure below.



4 Automata recognising Datalog trees

Given a Datalog program Π and a goal predicate Q, we construct a non deterministic tree automaton $A_{\Pi,Q} = (\Sigma, S, S_0, \rho, F)$, which recognizes the Datalog-Q-trees of Π. The construction is the following:
$\Sigma = \{s_1, ..., s_k\}$ is a finite non empty alphabet, where the symbol s_i, $1 \leq i \leq k$, corresponds to abstracted rule R_i and is understood to symbolize the application of the corresponding rule.
$S = \Sigma \bigcup \{true\}$ is the finite set of states, where $true$ is a new symbol designating acceptance.
$F = \{true\} \subset S$ is the set of accepting states.
$S_0 \subset S$ is the non empty set of initial states: S_0 is the set of symbols corresponding to abstracted rules with head the goal predicate Q.
$\rho : S \times \Sigma \rightarrow 2^{S^*}$ is the transition function, defined as follows:
- If $R_i : P\theta_i \leftarrow P_1\sigma_{(i,1)}, ..., P_n\sigma_{(i,n)}$ is an abstracted recursive rule then
$\rho((s_i, s_i)) = \{(s_1, ..., s_n) |\ R_1, ...R_n$ are abstracted rules with respective head predicates $P_1, ..., P_n\}$.
- If R_i is an abstracted initialization rule then $\rho((s_i, s_i)) = F = \{true\}$.

Theorem 4.1 Let Π be a Datalog program and Q be the goal predicate. Then the tree automaton $A_{\Pi,Q}$ recognizes the set of Datalog Q-trees of Π.

Example 4.2 The automaton corresponding to program Π of example 3.6 and to goal predicate P has the following features: $\Sigma = \{s_1, s_2, s_3, s_4, s_5\}$, $S_0 = \{s_1, s_2, s_4\}$ and the transition function ρ is such that $\rho((s_1, s_1)) = \{(s_1, s_3), (s_1, s_5), (s_2, s_3), (s_2, s_5), (s_4, s_3), (s_4, s_5)\}$, $\rho((s_2, s_2)) = \{(s_1), (s_2), (s_4)\}$, $\rho((s_3, s_3)) = \{true\}$, $\rho((s_4, s_4)) = \{(s_3), (s_5)\}$, $\rho((s_5, s_5)) = \{true\}$.

The complexity of the tree automaton $A_{\Pi,Q}$ can be measured with respect to the size of its state set S and of its transition function ρ, i.e. the number of transitions in ρ. It is easy to see that the tree automaton $A_{\Pi,Q}$ has exponential complexity: the size of the state set is linear but the number of transitions is exponential due to the fact that, for each rule, we consider all possible combinations of rules with head unifiable with a subgoal of the father rule. Notice though, that a Datalog tree (as defined here) contains exact information about the rules for each transition (hence contains also information about the EDB atoms involved in each transition). If we relax this requirement we can obtain the trees recognized by the automata defined in [Var94] which are still exponential; their exponentiality, though, comes from the fact that the size of the state set is exponential too (because all possible combinations of variables have to be considered). We believe that our technique can be used to define an automaton which accepts the trees in [Var94] and the complexity of the automaton being polynomial. We are currently working on that.

5 Conclusion

We defined here Datalog trees, a variable-free alternative to top-down evaluation trees, which appears to have many interesting features. First, Datalog trees offer an "abstracted" approach to the flow of variables, inherent to any resolution-based tree. The exact nature of the flow of variables has been revealed through the *transformation set* and the *constraint set* associated to each Datalog program. Indeed the resolution-based flow of variables has two aspects: the

renaming operation that occurs during unification is captured by the elements of the transformation set while bottom-up propagation of bindings (due to repetitions of variables in rule heads) is expressed by the elements of the constraint set. In usual approaches, presence of variables make necessary to consider different rule instances; thus evaluation trees as well as associated automata are defined up to isomorphism. In our context, we introduce *abstracted* program rules and this enables the precise (not up to isomorphism) definition of trees and automata. Moreover it is not hard (but still beyond the scope of the present paper) to map Datalog trees to the well-known evaluation trees mentioned in the introduction.

References

[Af97] F. Afrati. Bounded arity Datalog($\neq$) queries on graphs. *Journal of Computer and Systems Sciences*, vol 55 on ACM Conference "Principles of Database Systems,1997.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley Publishing Company, 1995.

[CGT90] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer-Verlag, 1990.

[FLS95] E. Foustoucos, K. Lellahi, and N. Spyratos. An algebraic semantics for Datalog. In *Journées Francophones de Programmation Logique*, 1995.

[Fou96] E. Foustoucos. *Une approche algébrique des méthodes d'Evaluation de Requêtes Datalog simplement linéaires.* Thèse de Doctorat, Université de Paris 11 (Orsay), 1996.

[GMN84] H. Gallaire, J. Minker, and J. M. Nicolas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 16(2):153–185, 1984.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[Min96] J. Minker. Logic and Databases: A 20 Year Retrospective. In Springer-Verlag LNCS Vol. 1154, 1-57, editor, *Proceedings of the Workshop on Logic in Databases*, 1996.

[Ull85] J.D. Ullman. Implementation of logical query languages for databases. *ACM Transactions on Database Systems*, 10(3):289–321, 1985.

[Rob65] J.A. Robinson. A machine-oriented logic based on the resolution principle. *J.Assoc.Comp.Mach*, 12(1):23–41, 1965.

[Ull89] J.D. Ullman. *Principles of Databases and Knowledge Base Systems*, volume I. Computer Science Press, 1989.

[Var92] M. Y. Vardi. Automata Theory for Database Theoreticians. *In J. D. Ullman, editor, Theoretical Studies in Computer Science*, pp. 153-180. Academic Press, 1992.

[Var94] M. Y. Vardi. Nontraditional applications of automata theory. *In Theoretical Aspects of Computer Software, Proc. Int. Symposium vol. 789 Lecture Notes in Computer Science (TACS94)*, pp. 575-597. Springer-Verlag, Berlin, 1994.

[Tho96] W. Thomas. Languages, Automata, and Logic. *Rep. 9607,Inst. f. Informatik u. Prakt. Math.*, University of Kiel, 1996.