Journal of Applied Logic 10 (2012) 368–409

Contents lists available at SciVerse ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Automata for the verification of monadic second-order graph properties

Bruno Courcelle [*,1], Irène Durand

Labri (CNRS), University of Bordeaux, 351 Cours de la Libération, F-33405 Talence cedex, France

A R T I C L E   I N F O

Article history:
Received 20 July 2011
Accepted 25 July 2011
Available online 27 July 2012

Keywords:
Graph algorithm
Automaton
Monadic second-order logic
Clique-width
Fixed-parameter tractability
Fly-automaton

A B S T R A C T

The model-checking problem for *monadic second-order logic* on graphs is *fixed-parameter tractable* with respect to tree-width and clique-width. The proof constructs finite automata from monadic second-order sentences. These automata recognize the terms over fixed finite signatures that define graphs satisfying the given sentences. However, this construction produces automata of hyper-exponential sizes, and is thus impossible to use in practice in many cases. To overcome this difficulty, we propose to specify the transitions of automata by programs instead of tables. Such automata are called *fly-automata*. By using them, we can check certain monadic second-order graph properties with limited quantifier alternation depth, that are nevertheless interesting for Graph Theory. We give explicit constructions of automata relative to graphs of bounded clique-width, and we report on experiments.

1. Introduction

It is well known from [5,6,10,12,16] that the model-checking problem for monadic second-order (MS) logic on graphs is fixed-parameter tractable (FPT) with respect to tree-width and clique-width. The proof uses certain graph decompositions: tree-decompositions for tree-width as parameter and decompositions in complete bipartite graphs for clique-width as parameter. Both types of decompositions are formalized by terms over finite sets of graph operations. The proof uses also finite automata, constructed from the MS sentences that express the properties to check. These automata recognize the terms that define graphs satisfying the given sentences.

There are two difficulties for turning this result into a usable algorithm. The first one is the *parsing problem* consisting in constructing an appropriate decomposition of the given graph. The second difficulty is due to the enormous sizes of the automata constructed from MS sentences. To address the latter, we propose to use *fly-automata*, i.e., automata whose transitions are specified by programs and not compiled in (huge) tables. We also present some tools that limit the number of states of the constructed automata: we construct "small" automata associated with some basic graph properties (and not with the atomic formulas) and we write formulas with *set terms* defined with the Boolean operations and the set variables.

In this article, we only consider the model-checking problem for monadic second-order sentences not using edge set quantifications. The relevant parameter is clique-width. Since for a class of graphs, bounded tree-width implies bounded clique-width, this approach also applies if tree-width is taken as parameter. Using tree-width as parameter allows to handle sentences written *with edge set quantifications*, but presents other difficulties (see [7]). Our objective is to implement the following theorem of [10] (we denote by F_k the finite set of graph operations that generates the graphs of clique-width at most k).

[*] Corresponding author.
E-mail addresses: courcell@labri.fr (B. Courcelle), idurand@labri.fr (I. Durand).
[1] Honorary member of Institut Universitaire de France.

1570-8683/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jal.2011.07.001

Theorem 1.

(1) *For every monadic second-order sentence φ and every integer k, one can construct a finite automaton recognizing the terms over F_k that denote graphs satisfying φ.*
(2) *Every monadic second-order graph property P can be checked in time $f_P(k) \cdot n^3$ for a simple directed or undirected graph with n vertices and of clique-width at most k.*

Assertion (1) gives an automaton that accepts or rejects a term in linear time in the size of the given term (a term over F_k for fixed k). For Assertion (2) we need to solve the parsing problem. Checking if a given graph has clique-width at most a given integer k is NP-complete if k is part of the input [14] but there exists a cubic approximation algorithm (see below Section 2.2). This algorithm takes time $g(k) \cdot n^3$ to construct a term of size $O(n)$ over $F_{h(k)}$ that denotes a given graph with n vertices and of clique-width at most k, where g and h are fixed functions. In other words the model-checking problem for monadic second-order logic on graphs is *fixed-parameter cubic* for a parameter consisting of a bound on the clique-width of the input graph and the MS sentence expressing the considered property.

Our only concern in this article is the construction of the automata of Assertion (1). They are constructed by induction on the structure of the input sentences (where universal quantifications are replaced by negations and existential quantifications). Those associated with the atomic formulas are easy to build and relatively "small". Products of automata are used for conjunction and disjunction. Complementation applied to deterministic automata is used for negation. A construction usually called "projection" is used for existential quantifications and introduces nondeterminism. It follows that determinization must be performed before each application of complementation. Since existential quantifications produce nondeterminism, quantifier alternation is the source of the hyper-exponential size of the constructed automata in the general case. This is actually unavoidable if one wants a construction taking as input arbitrary MS sentences (see, e.g., [17,29,31]). In order to overcome this difficulty, some authors focus their attention on particular problems instead of trying to implement the general theorem (see, e.g., [1,22,23,18,19]). We do not follow this route: we present some techniques that make the situation manageable for a large fragment of MS logic able to express interesting graph properties.

The article is organized as follows: Sections 2–4 review definitions about graphs, clique-width, automata and monadic second-order logic. Sections 4 and 5 constitute a tool box for the implementation of Theorem 1. Sections 5 and 6 detail the constructions of automata for the atomic formulas and for some basic graph properties. By using new atomic formulas expressing these properties and *Boolean set terms* (that do not cost much in terms of sizes of automata), we can express significant graph properties without quantifier alternation. Some constructions of Section 6 are somewhat complicated, and we prove their correctness. Section 7 defines fly-automata and some constructions concerning them. Section 8 reports on experiments with fly-automata. Section 9 is a conclusion.

2. Terms, graphs and clique-width

2.1. Terms and graphs

Definition 2 *(Terms and their syntactic trees).* A *functional signature* F is a set of *function symbols*, each being given together with a natural number called its *arity*: $\rho(f)$ denotes the arity of the symbol f. The set of *terms over* F is denoted by $T(F)$. A *language over* F is a subset of $T(F)$.

We now introduce some definitions relative to the internal structure of terms. A *position* of t is an occurrence of some symbol. We denote by $Pos(t)$ the set of positions of t and by $Pos(t, f)$ the set of occurrences in t of a symbol f. Hence $Pos(t) = \bigcup\{Pos(t, f) \mid f \in F\}$. The *size* $|t|$ of t is the cardinality of $Pos(t)$.

Terms will be written with commas and parentheses (that do not count in $|t|$). For example the term $t = f(g(h(a), b), c, g(g(b, c), c))$ has size 11. Positions will be designated by numbers corresponding to their ranks seen from left to right and starting at 1. In this example, $Pos(t, c) = \{6, 10, 11\}$. (They can be denoted in other ways, for instance by Dewey words, as in [4], our main reference for automata on terms.)

The *syntactic tree* of a term t is a rooted, labeled and ordered tree. Its set of nodes is $Pos(t)$. Each node u is labeled by the symbol f such that $u \in Pos(t, f)$ and it has an ordered sequence of $\rho(f)$ sons. The root ($root_t$) is the first position, and the occurrences of the nullary symbols are the leaves. (The terminology of rooted trees will thus be applied to terms, via their syntactic trees.) The partial order $\preccurlyeq_t$ on $Pos(t)$ is defined such that $u \preccurlyeq_t v$ if and only if $u = v$ or v is a proper ancestor of u. (Positions are integers but this order is not the usual order on integers.) We denote by t/u the *subterm of* t *issued from a node* u. In the above example, $t/2 = g(h(a), b)$. The *context of* u *in* t consists of the function symbols that occur at positions not below u: formally, it is the unique term over F with a unique occurrence of a variable x and such that t is equal to the substitution in c of t/u for x. In the above example, the context of $u = 2$ in t is $f(x, c, g(g(b, c), c))$.

Let H be a finite signature (possibly $H = F$), and $h : H \to F$ be an arity preserving mapping, i.e., such that $\rho(h(f)) = \rho(f)$ for every $f \in H$. For every $t \in T(H)$, we let $h(t) \in T(F)$ be the term obtained from t by replacing f by $h(f)$ at each of its occurrences. The mapping h on terms is called a *relabeling*.

Definition 3 *(Graphs).* All graphs are finite and simple. They can have loops. A graph G is identified with the relational structure $\langle V_G, edg_G\rangle$ where edg_G is a binary relation representing adjacency: $(x, y) \in edg_G$ if and only if there is a directed