Computing the Wadge Degree, the Lifschitz Degree, and the Rabin Index of a Regular Language of Infinite Words in Polynomial Time

Thomas Wilke[*] and Haiseung Yoo

Institut für Informatik und Prakt. Math., Christian-Albrechts-Universität zu Kiel, 24098 Kiel, Germany, E-mail: `tw@informatik.uni-kiel.d400.de`

Abstract. Based on a detailed graph theoretical analysis, Wagner's fundamental results of 1979 are turned into efficient algorithms to compute the Wadge degree, the Lifschitz degree, and the Rabin index of a regular ω-language: the two former can be computed in time $\mathcal{O}(f^2qb + k\log k)$ and the latter in time $\mathcal{O}(f^2qb)$ if the language is represented by a deterministic Muller automaton over an alphabet of cardinality b, with f accepting sets, q states, and k strongly connected components.

Formal languages are often compared via reductions: in recursion theory one compares formal languages using, e. g., truth-table reductions, in complexity theory formal languages are compared by, e. g., polynomial or log-space reductions, and in descriptive set theory continuous functions are used for the comparison of ω-languages.

In all these cases, given a formalism describing formal languages (such as Turing machines, grammars, automata, etc.), one can ask whether the reduction relation is decidable in the following sense: is there an algorithm that, on input of representations of languages X and Y, determines whether X is reducible to Y.

For ω-languages represented by Muller automata, in [9] Wagner answered this question affirmatively in case of continuous reductions and so-called 'synchronous' continuous reductions. Our main objective is to strengthen this by showing that the corresponding restrictions of the reduction relations are in fact decidable in polynomial time. This requires another thorough graph theoretical analysis of the loop structure of Muller automata, carried out below. We treat here only the asynchronous case; a glimpse at Wagner's paper [9] will suffice to realize the modifications necessary for the synchronous case.

A maximal class of formal languages that are pairwise reducible to each other is usually called degree. The degrees with respect to continuous and synchronous continuous reductions are called Wadge and Lifschitz degrees, respectively, see [7]. A such degree is called regular if it contains a regular ω-language.

In [9], K. W. Wagner introduced a naming system for regular Wadge and Lifschitz degrees. One important property of this system is that, given names α

[*] Supported by the ESPRIT BRA Working Group No. 6317, ASMICS 2.

and β for degrees D and E, respectively, one can determine in quadratic time whether the languages in D are reducible to the languages in E. Therefore, it is sufficient to establish a polynomial time algorithm that computes the name of the Wadge or Lifschitz degree of a regular ω-language given by a Muller automaton in order to prove that the respective restricted reduction relations are decidable in polynomial time.

We present an algorithm that, given a language L by a Muller automaton over an alphabet of cardinality b, with f accepting sets, q states, and k strongly connected components, computes the name of the Wadge degree of L in time $\mathcal{O}(f^2qb + k\log k)$. (Observe that always $k \leq q$ but f may be exponential in q.)

The algorithm is based on Wagner's discovery that the Wadge degree of a regular ω-language is determined by the loop structure—i. e., on the reachability relations and the inclusion relations between the accepting and rejecting loops—of any determinstic Muller automaton recognizing L. In other words, all Muller automata that recognize languages of one degree have the same loop structure.

As Wagner also discovered, all regular ω-languages with the same Rabin index (the least possible number of accepting pairs used in a Rabin automaton recognizing a language) form a set that is a union of Wadge degrees. Therefore our methods to analyse the loop structure of a Muller automaton can also be used to design an effecient algorithm computing the Rabin index of a regular ω-language. We present an algorithm running in time $\mathcal{O}(f^2qb)$, where f, q, and b are as above. This result contrasts with a recent result, see [3], that the problem of computing the Rabin index of a regular ω-language given by a deterministic Rabin or Streett automaton is $\mathcal{NP}$-complete. The reason for this is that the encoding of accepting loops using Rabin or Streett conditions may turn out more succinct than a mere enumeration (as in an equivalent Muller automaton on the same transition graph).

The paper is organized in six sections. In the first three sections we develop our algorithm computing the Rabin index of a regular ω-language, in the other three sections this algorithm is extended to our algorithm computing the Wadge degree. Sect. 1 introduces basic notions, in Sect. 2 the key lemma (Lemma 4) about the loop structure of a Muller automaton is stated and proved and the key procedure of all our algorithms is presented, and in Sect. 3 Wagner's result about the Rabin index and the results of Sect. 2 are combined to the desired algorithm. Sect. 4 and Sect. 5 review Wagner's results on the regular Wadge degrees, and in Sect. 6 we present our algorithm to compute the Wadge degree of a regular ω-language.

For Wagner's result concerning the Rabin index the reader is also referred to [8]. Wagner's approach to regular Wadge and Lifschitz degrees is automata theoretic,[2] a topological approach can be found in [4], an algebraic interpretation

[2] In [9], Wagner is mainly interested in (synchronous) reductions via functions definable by finite state machines. Yet in the last section of that paper he shows that, restricted to regular ω-languages, (synchronous) continuous reductions and (synchronous) finite-state machine definable reductions give rise to the same reduction relations.

is given in [1]. The article [6] is a general reference for ω-languages.

We would like to thank one of the referees for his or her comments which improved the paper significantly.

1 Alternating Chains and the Rabin Index

Throughout this paper, $\mathfrak{A}$ stands for a Muller automaton $(Q, q_0, \delta, \mathcal{F})$ over an alphabet denoted by B, and $\mathrm{L}(\mathfrak{A})$ stands for the language recognized by $\mathfrak{A}$.

The set V of nodes of the *transition graph* $\mathcal{G}_{\mathfrak{A}} = (V, E)$ of $\mathfrak{A}$ consists of the states of Q reachable from q_0, and an edge (q, q') belongs to the set E of edges of $\mathcal{G}_{\mathfrak{A}}$ if there exists a letter b such that $\delta(q, b) = q'$ and q is reachable from q_0. (Observe that the number of edges of $\mathcal{G}_{\mathfrak{A}}$ is bounded by $|Q||B|$.)

A *loop* in $\mathfrak{A}$ is a set $C \subseteq V$ such that the subgraph of $\mathcal{G}_{\mathfrak{A}}$ induced by C is strongly connected. The set $\mathcal{C}$ of all loops in $\mathfrak{A}$ is partitioned into the set $\mathcal{P} = \mathcal{C} \cap \mathcal{F}$ of *positive loops* and the set $\mathcal{N}$ of *negative loops*. The set of all maximal loops (with respect to set inclusion) is denoted by $\mathcal{M}$. (Notice that a maximal loop in $\mathfrak{A}$ is the same as a 'strongly connected component' of $\mathcal{G}_{\mathfrak{A}}$.) We shall use C, C_1, C', … for loops, P, P_1, P', … for positive loops, N, N_1, N', … for negative loops, and M, M_1, M', … for maximal loops.

An *alternating chain of length* n is of the form

$$C_1 \subset C_2 \subset C_3 \subset \ldots \subset C_n \,,$$

where ($C_i \in \mathcal{C}$ for every i with $1 \leq i \leq n$ and) $C_i \in \mathcal{P}$ iff $C_{i+1} \notin \mathcal{P}$ for every i with $1 \leq i < n$. A *positive alternating chain* starts with a positive loop, i.e. $C_1 \in \mathcal{P}$, and a *negative alternating chain* starts with a negative loop. If we say that an alternating chain is 'positive' or 'negative' we speak of the *sign* of the alternating chain, and sign stands for the set $\{+, -\}$.

The function $c_{\mathfrak{A}}: \mathcal{P} \to \mathbf{N}$ is defined as to map every positive loop P onto the length of a longest alternating chain starting with P.

Wagner proved in [8] (see also [9]) the following about the Rabin index of a regular ω-language.

Theorem 1 (Wagner). *The Rabin index of a regular ω-language recognized by a Muller automaton $\mathfrak{A}$ is given by the term*

$$\lfloor (\max\{c_{\mathfrak{A}}(P) \mid P \in \mathcal{P}\} + 1)/2 \rfloor \,. \tag{1}$$

Term (1) was the starting point of our search for an efficient algorithm computing the Rabin index of a regular ω-language. After a graph theoretical analysis of the loop structure of Muller automata we found an efficient way to compute $c_{\mathfrak{A}}$, and thus could establish an efficient algorithm for computing the Rabin index. The graph theoretical analysis is subject of the next section, while the algorithms computing $c_{\mathfrak{A}}$ and the Rabin index are presented in the next but one section.

2 Negative Loops In Between?

In this section we develop an efficient procedure that checks whether for two positive loops P_* and P^* with $P_* \subset P^*$ there is a negative loop N with $P_* \subset N \subset P^*$. This will be the key subroutine in the algorithms searching for long(est) alternating chains.

The problem with finding alternating chains is that sometimes the number of negative loops in an automaton happens to be exponential in the number of states and the number of positive loops (i.e. in the size of the automaton). Therefore one cannot simply compute all negative loops and search for alternating chains in the obvious way.

Let $\mathfrak{A}$ and $\mathcal{G}_{\mathfrak{A}}$ be as above. We say that a loop C_1 is *between* the loops C_0 and C_2 if $C_0 \subset C_1 \subset C_2$ holds.

If there is a loop N between P_* and P^* that is comparable with some positive loop P between P_* and P^* then either $P_* \subset N \subset P \subset P^*$ or $P_* \subset P \subset N \subset P^*$ holds. In this case the test whether there is a negative loop between P_* and P^* can be reduced to a 'smaller' one: is there a negative loop between P_* and P or between P and P^*, respectively? But what if there is no negative loop between P_* and P^* that is comparable with some positive loop in between?— This is what we examine first.

Let (*) be the following condition:

$$\left.\begin{array}{l} C_* \subset C^* \text{ are two loops and neither (2) nor (3) below hold for any } P \in \mathcal{P} \\ \text{and } N \in \mathcal{N}. \\ \qquad C_* \subset N \subset P \subset C^* \quad (2) \qquad\qquad C_* \subset P \subset N \subset C^* \quad (3) \end{array}\right\} (*)$$

Remark (complementation property). Assume (*). If $C_* \subset N \subset C^*$ and $C_* \subset P \subset C^*$, then $P \cup N = C^*$.

Proof. Assume $C = P \cup N$ is not C^*. If $C \in \mathcal{P}$, then $C_* \subset N \subset C \subset C^*$ contradicts (2). Otherwise $C_* \subset P \subset C \subset C^*$ contradicts (3). □

That is, two loops with complementary signs complement each other.

Let

$$D = \begin{cases} \bigcap\{P \mid C_* \subset P \subset C^*\}, & \text{if there is some } P \text{ with } C_* \subset P \subset C^*, \\ C^*, & \text{otherwise.} \end{cases} \qquad (4)$$

Lemma 2. *Assume (*) and $C_* \subset C \subset C^*$. Then $C \in \mathcal{P}$ iff $D \subseteq C$.*

Proof. For the non-trivial direction let $C \in \mathcal{N}$. If there is no P with $C_* \subset P \subset C^*$, we have $D \not\subseteq C$, since $D = C^*$ by definition. Otherwise, $P \cup C = C^*$ for every P with $C_* \subset P \subset C^*$ by the complementation property; hence $C^* \setminus C \subseteq P$ for every such P, whence

$$C^* \setminus C \subseteq D\,. \qquad (5)$$

If we had $D \subseteq C$, then (5) would imply $C^* \setminus C \subseteq C$, which, in turn, implies $C^* \subseteq C$. This is a contradiction to $C \subset C^*$, thus (5) cannot hold. Therefore $D \not\subseteq C$. □

If $\pi = q_1 \dots q_m$ is a path in $\mathcal{G}_{\mathfrak{A}}$, we write $||\pi||$ for the set $\{q_1, \dots, q_m\}$ and $|\pi|$ for m.

Definition 3. Let $C_* \subset C^*$. A *handle* is a non-empty path π satisfying the following conditions: $||\pi|| \subseteq C^* \setminus C_*$, and there exist $q, q' \in C_*$ such that $q\pi q'$ is also a path.

A handle is called *simple* if it is simple as a path.

Obviously, every minimal loop C with $C_* \subset C \subseteq C^*$ can be written as $C_* \cup ||\pi||$ where π is a simple handle. In presence of (*), this is true for negative and positive loops separately:

Remark. Assume (*).

1. If N is a minimal negative loop with $C_* \subset N \subseteq C^*$, then $N = C_* \cup ||\pi||$ for a suitable simple handle π. (A 'minimal negative loop N with $C_* \subset N \subseteq C^*$' is a minimal element of $\{N' \mid C_* \subset N \subseteq C^*\}$.)
2. If P is a minimal positive loop with $C_* \subset P \subseteq C^*$, then $N = C_* \cup ||\pi||$ for a suitable simple handle π.

Proof. 1) Consider a minimal loop C with $C_* \subset C \subseteq N$. This can be written as $C_* \cup ||\pi||$ for a suitable simple handle π. By (3), C is a negative loop, thus, by the minimality of N, $C = N$.

2) A dual argument applies. □

Lemma 4. *Assume (*) and the existence of a negative loop N between C_* and C^*. Then there is at most one minimal positive loop P between C_* and C^*.*

Proof. Assume for contradiction that there are two minimal positive loops P and P' of minimal cardinality between C_* and C^*. W.l.o.g. assume furthermore that N is minimal. Let P, P', and N be given by simple handles ϕ, ϕ', and ρ, respectively (see the above remark).

Since P and P' are distinct and minimal, there exist states q and q' with $q \in ||\phi|| \setminus ||\phi'||$ and $q' \in ||\phi'|| \setminus ||\phi||$. By Lemma 2, there exists a state $d \in D \setminus N$. This state is distinct from q and q', because otherwise $D \subseteq P$ or $D \subseteq P'$ would not hold (Lemma 2). Since $N \cup P = C^*$ and $N \cup P' = C^*$ (complementation property) we also have $q' \in N$ and $q \in N$.

According to the order in which q, q', and d occur in ϕ, ϕ', and ρ, we can write ϕ, ϕ', and ρ, respectively, as follows, where in each line the respective two possible choices are listed.

$$\phi = \phi_0 d \phi_1 q \phi_2 \quad (6) \qquad \phi = \phi_0 q \phi_1 d \phi_2 \quad (9)$$

$$\phi' = \phi'_0 d \phi'_1 q' \phi'_2 \quad (7) \qquad \phi' = \phi'_0 q' \phi'_1 d \phi'_2 \quad (10)$$

$$\rho = \rho_0 q \rho_1 q' \rho_2 \quad (8) \qquad \rho = \rho_0 q' \rho_1 q \rho_2 \quad (11)$$

(Observe that $\{d, q, q'\} \cap (\bigcup_{i=0}^2 ||\phi_i|| \cup \bigcup_{i=0}^2 ||\phi'_i|| \cup \bigcup_{i=0}^2 ||\rho_i||) = \emptyset$ because of the minimality of P, P', and N.) We will see that every of the eight possible combinations leads to a contradiction.

Two arguments will be used again and again in what follows. For reference, we state them before carrying on with the proof.

Let C be a loop such that $C_* \subset C \subseteq C^*$ holds.

(i) If $d \notin C$, then $C \notin \mathcal{P}$.
(ii) If $q \notin C$ or $q' \notin C$, then $C \notin \mathcal{N}$.

The first claim is true, because d belongs to the intersection defined in (4). The second claim is true because of the complementation property: if $q \notin C$, then $q \notin C \cup P'$, thus $C \cup P \subset C^*$, hence $C \notin \mathcal{N}$ by the complementation property. A symmetric argument applies if $q' \notin C$.

Since the paths ϕ, ϕ', and ρ have points in common, it is possible to compose new paths (and handles) from suitable segments. Which paths can be built depends on the equations that hold. For instance, if we have (9) and (11), then $\phi_0 q \rho_2$ is a path, even a handle.

If one of

$$\phi_0 q \rho_2,\ \phi_0' q' \rho_2,\ \rho_0 q \phi_2, \text{ or } \rho_0 q' \phi_2' \tag{12}$$

is a path, the desired contradiction is easy to obtain. We demonstrate this for $\phi_0 q \rho_2$, the other cases are similar.

Consider $\pi = \phi_0 q \rho_2$, which is a handle. On the one hand, the loop $C_* \cup ||\pi||$ is not positive, because d does not belong to it (cf. (i)). On the other hand, q' does not belong to $C_* \cup ||\pi||$, hence $C_* \cup ||\pi||$ is no negative loop (cf. (ii)).

Only if

1. (9), (7), and (8) hold, or if
2. (6), (10), and (11) hold,

one cannot immediately construct one of the paths enumerated in (12). The two cases are symmetric, so we deal only with the first case.

Consider the handle $\pi = \phi_0' d \phi_2$. Since q does not belong to $C_* \cup ||\pi||$, this loop is positive (cf. (ii)). From the minimality of P' we know $|\phi_2| \geq |\phi_1' q' \phi_2'|$. (Otherwise the cardinality of $C_* \cup ||\pi||$ would be strictly smaller than the cardinality of P'.) Since q' does not belong to $||\phi_2||$, there exists a state $p \in ||\phi_2|| \setminus ||\phi_1' q' \phi_2'||$, say $\phi_2 = \psi_0 p \psi_1$.

We claim that p does not occur in ϕ_0' (which implies $p \notin ||\phi'||$ since $p \notin ||\phi_1' q' \phi_2'||$ by definition). For contradiction, assume p occurs in ϕ_0', say $\phi_0' = \delta_0 p \delta_1$. Then $\pi' = \delta_0 p \psi_1$ is a handle. But the loop $C_* \cup ||\pi'||$ is neither positive, since it does not contain d (cf. (i)), nor negative, since it does not contain q (cf. (ii)).

We have $p \notin ||\phi'||$, and $C_* \cup ||\phi_0' d \psi_0 p \psi_1||$ is a positive loop, since q does not belong to it (cf. (ii)). Therefore P and $\phi = \phi_0 q \phi_1 d \phi_2$ can be replaced by $C_* \cup ||\phi_0' d \psi_0 p \psi_1||$ and $\phi_0' d \psi_0 p \psi_1$, respectively. We encounter a case that leads immediately to a contradiction as above. □

By symmetry, we have:

Corollary 5. *Assume (*). If there are positive and negative loops between C_* and C^*, then there is a unique minimal (i. e. smallest) positive loop and a unique minimal (i. e. smallest) negative loop between C_* and C^*.*

We now can design an efficient procedure that takes loops $P_* \subset P^*$ as arguments and gives back TRUE if there exists N such that $P_* \subset N \subset P^*$ and FALSE otherwise, provided (*) holds (with $C_* = P_*$ and $C^* = P^*$):

1. Find a minimal loop C with $P_* \subset C \subseteq P^*$.
2. If $C \notin \mathcal{P}$, then return TRUE. (C is negative and between P_* and P^*.)
3. If $C = P^*$, then return FALSE. (There is no loop between P_* and P^*, in particular, no negative one.)
4. Search for a minimal loop C' with $P_* \subset C' \subseteq P^*$ and $C \neq C'$.
5. If there exists no such C', then return FALSE. (There is only one minimal loop between P_* and P^*, namely C, and this is positive. Thus, by (*), there is no negative loop between P_* and P^*.)
6. If there exists such a C' and
   (a) if $C' \notin \mathcal{P}$, then return TRUE (C' is negative and between P_* and P^*),
   (b) if $C' \in \mathcal{P}$, then return FALSE. (There is no unique minimal positive loop between P_* and P^*, thus there exists no negative loop between P_* and P^* by Lemma 4.)

To complete the description of the procedure we have to explain how steps 1 and 4 can be implemented.

Step 1. Since at least one minimal loop C with $P_* \subset C \subseteq P^*$ is given by a shortest handle, it is sufficient to search for a shortest handle π, and to set $C = P_* \cup ||\pi||$. We find such a handle using a breadth-first strategy in the subgraph of $\mathcal{G}_{\mathfrak{A}}$ induced by P^*: we search for a shortest non-empty path from P_* to P_* and leaving P_*. This takes time $\mathcal{O}(|P^*||B|)$.

Step 4. The situation is a bit more involved but essentially the same idea works; we search for simple paths satisfying certain conditions.

We obtain the following upper bound for the running time of the entire test.

Lemma 6. *Given a Muller automaton $\mathfrak{A}$ and two positive loops $P_* \subset P^*$ such that (*) holds (with $C_* = P_*$ and $C^* = P^*$), the above procedure checks in time $\mathcal{O}(|Q|(|B| + |\mathcal{F}|))$ whether there exists a negative loop between P_* and P^*.*

This bound takes also into account that we have to construct $\mathcal{G}_{\mathfrak{A}}$ and a search tree for the elements of $\mathcal{P}$ (in order to be able to perform a test as in step 2 in time $|Q|$).

3 Computing the Rabin Index

As pointed out at the end of Sect. 1, in order to compute the Rabin index we need to compute (the maximum value of) the function $c_{\mathfrak{A}}$. The rules that allow us to determine $c_{\mathfrak{A}}$ inductively are summed up in the following remark.

Remark. 1. If P is a maximal positive loop, then $c_{\mathfrak{A}}(P) = \begin{cases} 2\,, & \text{if } P \notin \mathcal{M}, \\ 1\,, & \text{otherwise.} \end{cases}$

2. If P is not a maximal positive loop, if $k = \max\{c_{\mathfrak{A}}(P') \mid P \subset P'\}$, and if $\mathcal{P}' = \{P' \mid P \subset P' \wedge c_{\mathfrak{A}}(P') = k\}$, then

$$c_{\mathfrak{A}}(P) = \begin{cases} k+2\,, & \text{if there are } N \text{ and } P' \in \mathcal{P}' \text{ such that } P \subset N \subset P', \\ k\,, & \text{otherwise.} \end{cases}$$

This remark motivates and proofs the correctness of the following procedure computing $c_{\mathfrak{A}}$:

1. For every maximal $P \in \mathcal{P}$, if $P \in \mathcal{M}$, then let $c_{\mathfrak{A}}(P) = 1$, else $c_{\mathfrak{A}}(P) = 2$.
2. For every non-maximal $P \in \mathcal{P}$ in non-increasing order:
   (a) Let $k = \max\{c_{\mathfrak{A}}(P') \mid P \subset P'\}$, $\mathcal{P}' = \{P' \mid P \subset P' \wedge c_{\mathfrak{A}}(P') = k\}$, and $c_{\mathfrak{A}}(P) = k$.
   (b) For every $P' \in \mathcal{P}'$ in non-decreasing order, if there is a negative loop between P and P', then let $c_{\mathfrak{A}}(P) = k+2$ and exit this for-loop.

In step 2(b), it is essential to test the elements of $\mathcal{P}'$ in non-decreasing order because otherwise (*) could not be guaranteed (with $C_* = P$ and $C^* = P'$) when launching the test whether there exists a negative loop between P and P'.

Taking Lemma 6 into account we get:

Lemma 7. *Given a Muller automaton $\mathfrak{A} = (Q, q_0, \delta, \mathcal{F})$ over an alphabet B, the above procedure computes in time $\mathcal{O}(|\mathcal{F}|^2|Q||B|)$ the function $c_{\mathfrak{A}}$.*

The upper bound for the running time also takes into account the time we need to compute the set $\mathcal{M}$ (which is $\mathcal{O}(|Q||B|)$ by Tarjan's algorithm, see [5]) and the inclusion relation on $\mathcal{P}$ (which is $\mathcal{O}(|\mathcal{F}|^2|Q|)$).

As a consequence of Lemma 7 and Theorem 1, we obtain our first theorem:

Theorem 8. *The Rabin index of a regular ω-language given by a Muller automaton $\mathfrak{A} = (Q, q_0, \delta, \mathcal{F})$ over an alphabet B can be computed in time $\mathcal{O}(|\mathcal{F}|^2|Q||B|)$.*

4 Regular Wadge Degrees and Wagner's Naming System

The set B^ω of all ω-words over an alphabet B is turned into a metric space by introducing the distance function d with $d(\alpha, \beta) = 2^{-\min\{i \mid \alpha(i) \neq \beta(i)\}}$ for distinct ω-words α and β. A set $L \subset B^\omega$ is *Wadge reducible* to a set $M \subseteq B^\omega$, in symbols $L \leq_W M$, if L is the inverse image of M under a continuous function $B^\omega \to B^\omega$. The $\leq_W$-relation is an equivalence relation, and each equivalence class is called a *Wadge degree*. The relation $\leq_W$ extends in a natural way to the degrees. A Wadge degree is called *regular* if it contains a regular ω-language.

As mentioned in the introduction in [9] Wagner investigated the structure of the set of all regular Wadge degrees (ordered by $\leq_W$). In a unique way he denoted each degree by an expression, henceforth called *name*, of the form $E_{i_1}^{j_1} \ldots E_{i_{s-1}}^{j_{s-1}} x_{i_s}^{j_s}$, where x is either of the symbols C, D, or E, $s \geq 1$ and $i_1 > \ldots > i_s$ and, in case $x = E$, $i_s = 1$. In this paper, the name of the Wadge degree of a regular ω-language L is denoted by $W(L)$.

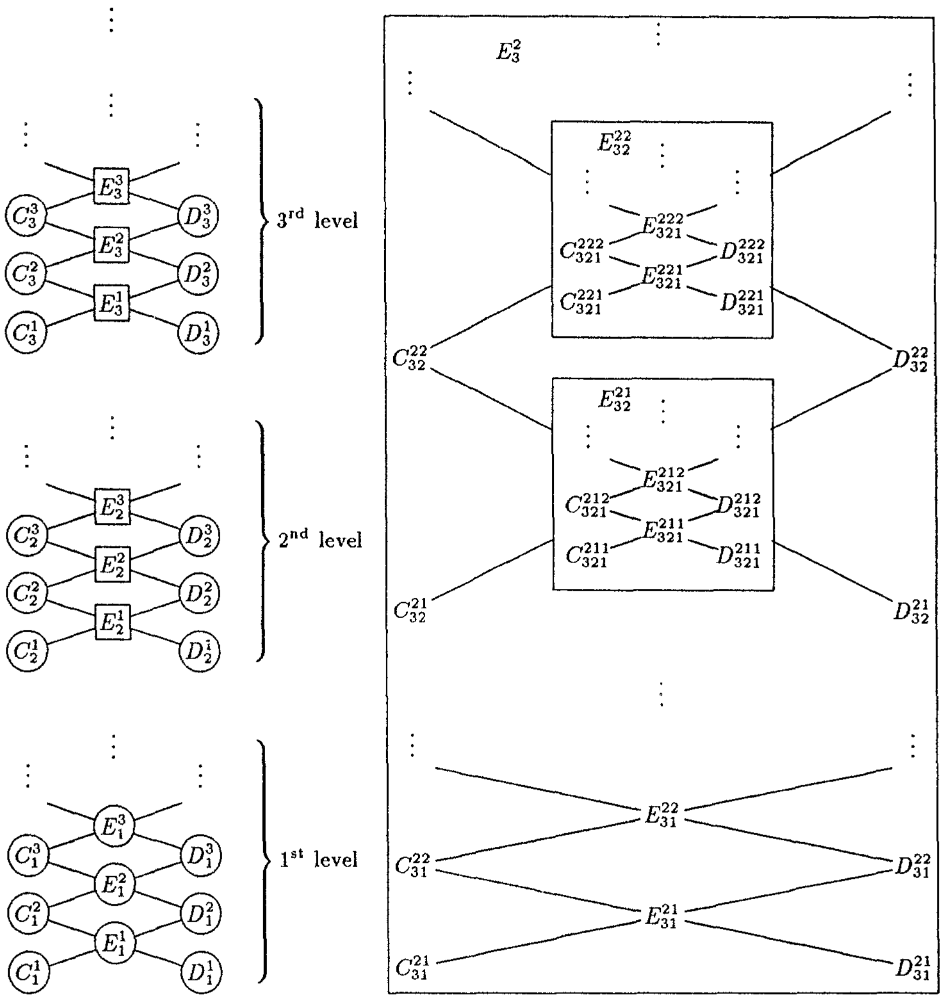

Fig. 1. The structure of the ordering of the regular Wadge degrees

Wagner's naming system is chosen in such a way that the $\leq_W$-relation can easily be read off: for regular degrees S and T with names $E^{j_1}_{i_1}\dots E^{j_{s-1}}_{i_{s-1}}x^{j_s}_{i_s}$ and $E^{l_1}_{k_1}\dots E^{l_{t-1}}_{k_{t-1}}y^{l_t}_{k_t}$, the relation $S \leq_W T$ holds iff there is an index u such that $i_v = k_v$ and $j_v = l_v$ for v with $1 \leq v \leq u$, and either (a) $s = u \leq t$, and $y = E$ or $x = y$, or (b) $u < s$ and $u < t$ and $i_{u+1} < k_{u+1}$.

The coarse structure of the set of all regular Wadge degrees can be presented graphically as in the left part of Fig. 1. For notational convenience we write $x^{j_1 j_2 \dots j_s}_{i_1 i_2 \dots i_s}$ for $E^{j_1}_{i_1}\dots E^{j_{s-1}}_{i_{s-1}}x^{j_s}_{i_s}$, and if $i_s > 1$ the union of all degrees whose names have $E_{i_1}^{j_1} \dots E_{i_s}^{j_s}$ as prefix is denoted by $E_{i_1 i_2 \dots i_s}^{j_1 j_2 \dots j_s}$. Circles represent one and only one degree, boxes stand for a union of several degrees. The first lower index of a name of a degree divides the entire hierarchy into an infinite number of levels. Every box E_m^n of the mth level ($m > 1$) is structured exactly as the part of the hierarchy that consists of the levels $1, \dots, m-1$. The corresponding circles and boxes in E_m^n are denoted by C_{mi}^{nj}, D_{mi}^{nj}, E_{mi}^{nj}, respectively. Again, if $i > 1$, then E_{mi}^{nj} is divided into further degrees by the same procedure. This gives a 'recursive' structure (but leading only to finite descending chains). For instance, the structure of E_3^2 is as depicted in the right part of Fig. 1.

5 Wagner's Naming Procedure

In the previous section, we presented and explained the structure of the regular Wadge degrees and how Wagner named them. In this section, we will explain how, given a Muller automaton $\mathfrak{A}$, one can determine the name of the degree of L($\mathfrak{A}$). In principle, we follow [9]. By introducing the notion of 'condensed graph', we slightly change the style of presentation. This is useful from an algorithmic point of view.

We first come back to alternating chains, and assume a Muller automaton $\mathfrak{A} = (Q, q_0, \delta, \mathcal{F})$ over an alphabet B to be given. If $C_1 \subseteq \dots \subseteq C_n$ is an alternating chain in $\mathfrak{A}$, then all the loops C_i with $1 \leq i \leq n$ belong to the same maximal loop. If this loop is M then we say that *the chain is in* M.

With $\mathfrak{A}$, we associate the *length function* $l_{\mathfrak{A}}: \mathcal{M} \rightarrow \mathrm{N}$ that maps every maximal loop M onto the length of a longest alternating chain in M.

Let M be a fixed maximal loop of $\mathfrak{A}$. In every longest alternating chain in M one can replace the last element by M itself. Therefore the signs of all longest alternating chains in M are the same, and the *sign function* $s_{\mathfrak{A}}: \mathcal{M} \rightarrow$ sign that maps every maximal loop M on the sign of the longest alternating chain in M is well-defined.

A *condensed graph* is a quintuple (V, E, l, s) such that (V, E) is a directed acyclic graph and $l: V \rightarrow \mathrm{N}$ and $s: V \rightarrow$ sign are labellings of vertices. We say that a node v is positive if $s(v) = +$ and negative otherwise.

The *condensed graph associated with the Muller automaton* $\mathfrak{A}$ is the condensed graph $\mathcal{K}_{\mathfrak{A}} = (\mathcal{M}, E, l_{\mathfrak{A}}, s_{\mathfrak{A}})$, where E contains an edge from M to M' iff there is a path from M to M' in $\mathcal{G}_{\mathfrak{A}}$ that does not contain any state belonging to one of the other maximal loops (i.e. when M' is an immediate successor of M).

In the following, let $G = (V, E, l, s)$ be a condensed graph. We define m_G^+ to be the maximum of $\{l(M) \mid s(M) = +\} \cup \{0\}$, and m_G^- analogously. That is, if $G = \mathcal{K}_{\mathfrak{A}}$ then m_G^+ is the length of a longest positive alternating chain in $\mathfrak{A}$, and m_G^- stands for the length of a longest negative alternating chain in $\mathfrak{A}$. We write m_G for the maximum of m_G^+ and m_G^-.

A *top node* in G is a node M such that $l(M) = m_G$. An *alternating superchain* is a sequence $C_1, \dots, C_n$ of top nodes where for every i with $1 \leq i < n$ the node C_{i+1} is reachable from C_i and C_i is a positive top node iff C_{i+1} is not. The alternating superchain is called positive if C_1 is a positive top node and negative otherwise.

We define n_G^+ to be the maximum length of a positive superchain if there is some and 0 otherwise. The number n_G^- is defined correspondingly. The maximum of both numbers is denoted by n_G. A *longest superchain* is an alternating superchain of length n_G. The first node in a positive longest superchain is called a *positive base node* and the notion of a *negative base node* is defined similarly.

The condensed graph G is called *non-prime* if $n_G = n_G^- = n_G^+$ and *prime* otherwise.

For every condensed graph G with $m = m_G$ and $n = n_G$, we define its *type*, denoted by t_G. If G is non-prime, then $t_G = E_m^n$, if G is prime and $n_G^+ > n_G^-$, then $t_G = C_m^n$, else, if G is prime and $n_G^- > n_G^+$, then $t_G = D_m^n$. The set type is defined to be the set $\{x_m^n \mid x \in \{C, D, E\} \wedge m, n \in \mathrm{N}\}$. (So every name of a Wadge degree can be viewed as a string over type satisfying certain conditions.)

For a non-prime condensed graph $G = (V, E, l, s)$ we define its *derivative* ∂G. It is the condensed graph (V', E', l', s'), where V' is the set of nodes from which both a positive and a negative base node are reachable, $E' = E \cap (V' \times V')$, and l' and s' are the restrictions of l and s to V'.

With every condensed graph G, we associate a name $W(G)$. If $G = \mathcal{K}_{\mathfrak{A}}$ is the condensed graph of a Muller automaton $\mathfrak{A}$ recognizing the ω-language L, then $W(G)$ will be equal to $W(L)$.

The name $\alpha = W(G)$ is defined by the following procedure, also called *Wagner's naming procedure*, which iteratively constructs $W(G)$ by essentially concatenating the types of G, ∂G, $\partial^2 G$, ... :

```
1. Let $\alpha = \epsilon$, $i = 0$.
2. Do forever:
   (a) Let $i = i + 1$.
   (b) If $G$ is empty then return $\alpha E_1^1$.
   (c) If $G$ is prime then return $\alpha t_G$.
   (d) If $G$ is non-prime, $m_G = 1$, and $i = 1$ then return $t_G$.
   (e) If $G$ is non-prime, $m_G = 1$, and $i > 1$ then return $\alpha E_{m_G}^{n_G+1}$.
   (f) If $G$ is non-prime and $m_G > 1$ then let $\alpha = \alpha t_G$ and $G = \partial G$.
```

Observe that only step 2(f) leads to a new iteration of the for loop.

The correctness of the naming procedure is stated in the following theorem.

Theorem 9 (Wagner). *For every Muller automaton $\mathfrak{A}$, $W(\mathrm{L}(\mathfrak{A})) = W(\mathcal{K}_{\mathfrak{A}})$.*

In particular, since the above naming procedure is effective, it follows that the name of the Wadge degree of a regular ω-language can be computed effectively.

In the next section, we will describe how the above procedure can be implemented efficiently.

6 Computing the Wadge degree

Following Theorem 9 our algorithm that, given a Muller automaton $\mathfrak{A}$, computes the name of the Wadge degree of $\mathrm{L}(\mathfrak{A})$ consists of three steps:

1. Build $G = \mathcal{K}_{\mathfrak{A}}$.
2. Determine $\alpha = W(G)$.
3. Output α.

Step 3 is straightforward. The other two steps are treated in the following two subsections.

6.1 Step 1

We first compute the transition graph $\mathcal{G}_{\mathfrak{A}}$ in an obvious way. Next, using Tarjan's algorithm, we compute the vertex set of $\mathcal{K}_{\mathfrak{A}}$ (i.e. the set $\mathcal{M}$ of all maximal loops of $\mathcal{G}_{\mathfrak{A}}$) and the edge relation.

In order to determine $l_{\mathfrak{A}}$ and $s_{\mathfrak{A}}$ we first compute $c_{\mathfrak{A}}$ as described in Sect. 3. After that, for each $M \in \mathcal{M}$, we partition the set $\mathcal{R}$ of all minimal positive loops in M into the sets $\mathcal{S}$ and $\mathcal{T}$ according to whether a loop is a superset of a negative loop or not. Using the following expressions we then determine $l_{\mathfrak{A}}(M)$ and $s_{\mathfrak{A}}(M)$.

$$l_{\mathfrak{A}}(M) = \max(\{c_{\mathfrak{A}}(P) + 1 \mid P \in \mathcal{S}\} \cup \{c_{\mathfrak{A}}(P) \mid P \in \mathcal{T}\}) \tag{13}$$

$$s_{\mathfrak{A}}(M) = \begin{cases} +\,, & \text{if } l_{\mathfrak{A}}(M) \equiv 1 \pmod 2 \text{ and } M \in \mathcal{P}, \text{ or} \\ & \text{if } l_{\mathfrak{A}}(M) \equiv 0 \pmod 2 \text{ and } M \notin P, \\ -\,, & \text{otherwise} \end{cases} \tag{14}$$

To compute the sets $\mathcal{R}$, we need time $\mathcal{O}(|\mathcal{F}|^2)$. To partition one set $\mathcal{R}$ we need to check whether in a given connected subset of a graph there exists a smaller connected set. This can be done using a breadth-first search. (There exists no smaller connected subset iff the breadth-first search tree is a path and there are no back edges except for a back edge from the last point of the path to its beginning.)

The upper bound for the computation of $c_{\mathfrak{A}}$ (see Lemma 7) is $\mathcal{O}(|\mathcal{F}|^2|Q||B|)$ and dominates the time needed to compute $l_{\mathfrak{A}}$ and $s_{\mathfrak{A}}$, starting from $c_{\mathfrak{A}}$ as just described, and to perform Tarjan's algorithm.

Remark. Step 1 is performed in time $\mathcal{O}(|\mathcal{F}|^2|Q||B|)$.

6.2 Step 2

The execution of step 2 follows Wagner's naming procedure (see Sect. 5). To avoid misunderstanding, the two steps of that procedure are called *phases* hereafter. Apart from α and i, variables LEVEL, TYPE, TO-VISIT, and MAX-NODES are initialized in phase 1. These additional variables are used and updated in phase 2.

The current derivative (i.e. G) is represented by the array LEVEL: $\mathcal{M} \to \mathbf{N}$ according to the following convention (invariant).

($\ast$) Before each execution of the for loop (i.e. at the beginning of 2(a)) G contains the nodes M with LEVEL$[M] \geq i$. The variable MAX-NODES contains a list of all maximal nodes of G (viewed as a partial order).

Recall that a derivative of a condensed graph is a subgraph induced by a certain set of nodes. Therefore the current derivative is fully determined by its set of nodes.

Phase 1: Initialization. The array TO-VISIT is initialized to PERHAPS, and LEVEL to the number of nodes of G. (The latter is in accordance with ($\overset{*}{\underset{*}{}}$).)

Each vertex M of the condensed graph G is labelled with the type of the graph which is obtained from G by removing all vertices which M is not reachable from (see below). The labels are stored in the array TYPE. Furthermore, a list of the maximal nodes in G is produced and stored in MAX-NODES.

For the computation of TYPE we introduce two functions:

$$\sqcup\colon \text{type} \times \text{type} \longrightarrow \text{type} \qquad \text{and} \qquad \lambda\colon \text{type} \times \mathbf{N} \times \text{sign} \longrightarrow \text{type}.$$

The function $\sqcup$ is defined in such a way that if G_0 and G_1 are condensed graphs then $t_{G_0} \sqcup t_{G_1}$ is the type of the disjoint union of G_0 and G_1:

$$x_m^n \sqcup y_k^l = y_k^l \sqcup x_m^n = \begin{cases} x_m^n\,, & \text{if } m > k, \text{ or} \\ & \text{if } m = k \text{ and } n > l, \text{ or} \\ & \text{if } m = k \text{ and } n = l \text{ and } x = y, \\ E_m^n\,, & \text{otherwise.} \end{cases}$$

Obviously, $\sqcup$ is an associative and commutative operation.

The function λ is defined such that if $G_0 = (V, E, l, s)$ is a condensed graph with one unique maximal node v and if G_0 is obtained from a graph G_1 by adding v then $\lambda(t_{G_1}, l(v), s(v)) = t_{G_0}$. If $m' > m$ then

$$\lambda(x_m^n, m', s) = \begin{cases} C_{m'}^1\,, \text{ if } s = +, \\ D_{m'}^1\,, \text{ if } s = -. \end{cases}$$

If $m' < m$ then $\lambda(x_m^n, m', s) = x_m^n$. Furthermore

$$\lambda(x_m^n, m, s) = \begin{cases} x_m^{n+1}\,, & \text{if } s = -,\ x = C, \text{ and } n \equiv 1 \pmod 2, \text{ or} \\ & \text{if } s = +,\ x = D, \text{ and } n \equiv 1 \pmod 2, \\ C_m^{n+1}\,, & \text{if } s = +,\ x = E, \text{ and } n \equiv 0 \pmod 2, \text{ or} \\ & \text{if } s = -,\ x = E, \text{ and } n \equiv 1 \pmod 2, \\ D_m^{n+1}\,, & \text{if } s = +,\ x = E, \text{ and } n \equiv 1 \pmod 2, \text{ or} \\ & \text{if } s = -,\ x = E, \text{ and } n \equiv 0 \pmod 2, \\ x_m^n\,, & \text{otherwise,} \end{cases}$$

for every possible choice of x, m, s, and n.

Since both $\sqcup$ and λ are defined by simple case distinctions, each application of them in a procedure will take constant time.

Now, using $\sqcup$ and λ we can describe a rule that allows us to compute the values of the array TYPE: If $M_1, \ldots, M_r$ is the list of predecessors of node M, then

$$\text{TYPE}(M) = \lambda(\text{TYPE}(M_1) \sqcup \ldots \sqcup \text{TYPE}(M_r), l(M), s(M))\,.$$

Therefore, the types of all nodes can be computed using a simple for loop treating the nodes in a topological ordering (to make sure that the values of the predecessors are already known). For a minimal node M, we have $\mathrm{TYPE}(M) = C^1_{l_{\mathfrak{A}}(M)}$ if $s(M) = +$ and $\mathrm{TYPE}(M) = D^1_{l_{\mathfrak{A}}(M)}$ otherwise. In the same for loop one can also compute the list MAX-NODES for the entire graph. A topological ordering of the nodes of G can be computed in linear time (see [2]).

So we get the following for the complexity of phase 1.

Remark. Phase 1 is performed in time $\mathcal{O}(|\mathcal{M}| + |E|)$, where E is the number of edges in $\mathcal{K}_{\mathfrak{A}}$.

Phase 2: Execution of the For Loop. There are two points to be discussed here: I) how t_G (needed in 2(c), (d), and (f)) is determined, and II) how ∂G (needed in 2(f)) can be computed, if G is non-prime—the other parts of phase 2 are simple tests or assignments.

ad I). The type of G is determined by combining the TYPE values of the nodes in MAX-NODES using $\sqcup$; this takes time linear in the length of MAX-NODES.

ad II). We are in 2(f). So LEVEL and MAX-NODES have to be updated to represent the derivative of $G = (V, E, l, s)$ in the sense of $(^*_*)$. We use two additional arrays POS-S-CHAIN and NEG-S-CHAIN, and a heap H. The arrays POS-S-CHAIN and NEG-S-CHAIN store for every $M \in V$ the length of the longest positive and negative superchain, respectively, that is reachable from M in G. This information is used to determine the nodes that have to be deleted in order to obtain the desired derivative, i.e., the nodes from which no positive or no negative base loop is reachable. On the heap H, the candidates for deletion are stored, the node being the greatest in a topological ordering of $\mathcal{K}_{\mathfrak{A}}$ on top of H. The array TO-VISIT records which nodes are already on the heap and which of them are to be disregarded (although they are on the heap).

For every element M of MAX-NODES, we set LEVEL$[M]$ to $i-1$, put all its predecessors on the heap H, and set TO-VISIT$[M']$ to YES for each such predecessor. If $l(M) = m_G$ and $s(M) = +$, the variable POS-S-CHAIN$[M]$ is set to 1 and POS-S-CHAIN$[M]$ to 0. If $l(M) = m_G$ and $s(M) = -$, the variable NEG-S-CHAIN$[M]$ is set to 1 and POS-S-CHAIN$[M]$ to 0. In all other cases, 0 is assigned to POS-S-CHAIN$[M]$ and NEG-S-CHAIN$[M]$. The list MAX-NODES is emptied.

In a while loop we proceed until the heap H is empty. In every iteration, a node M is extracted from H and processed as follows. If TO-VISIT$[M]$ is not YES, nothing happens. Otherwise, POS-S-CHAIN$[M]$ and NEG-S-CHAIN$[M]$ are determined. This is done by combining the corresponding values of the successors of M in G, the value $l(M)$, and the value $s(M)$. If POS-S-CHAIN$[M] < n_G$ or NEG-S-CHAIN$[M] < n_G$, LEVEL$[M]$ is set to $i-1$, i.e., M is deleted, and every predecessor M' of M is added to the heap, provided TO-VISIT$[M'] =$ PERHAPS. In the other case, if POS-S-CHAIN$[M] =$ NEG-S-CHAIN$[M] = n_G$, M is put into MAX-NODES and TO-VISIT$[M']$ is set to NO for every predecessor M' of M. After exit of the while loop, TO-VISIT is reset to PERHAPS and H is emptied.

Remark. Phase 2 is performed in time $\mathcal{O}(|E| + |\mathcal{M}| \log |\mathcal{M}|)$, where E is the number of edges in $\mathcal{K}_{\mathfrak{A}}$.

The factor of $\log |\mathcal{M}|$ reflects the time that is needed to maintain the heap H.

Adding up the running times for step 1, phase 1, and phase 2, we finally obtain:

Theorem 10. *The name of the Wadge degree of a regular ω-language given by a Muller automaton $\mathfrak{A} = (Q, q_0, \delta, \mathcal{F})$ can be computed in time $\mathcal{O}(|\mathcal{F}|^2|Q||B| + |\mathcal{M}| \log |\mathcal{M}|)$, where $\mathcal{M}$ is the set of all maximal loops in $\mathfrak{A}$.*

References

1. O. CARTON. "Mots Infinis, ω-Semigroupes et Topologie". PhD thesis, Université Paris 7, France (1993).
2. D. E. KNUTH. "Fundamental Algorithms", vol. 1. Addison-Wesley (1968). Second edition 1973.
3. S. C. KRISHNAN, A. PURI, AND R. K. BRAYTON. Structural complexity of ω-automata. In "STACS 95: 12th Annual Symposium on Theoretical Aspects of Computer Science", München (1995), Lecture Notes in Computer Science. Springer-Verlag. To appear.
4. V. SELIVANOV. Fine hierarchy of regular ω-languages. This volume.
5. R. E. TARJAN. Depth first search and linear graphs. *SIAM J. Comput.* **1**(2), 146–160 (1972).
6. W. THOMAS. Automata on infinite objects. In J. VAN LEEUWEN, editor, "Handbook of Theoretical Computer Science", vol. B: Formal Methods and Semantics, pp. 134–191. Elsevier Science Publishers B. V. (1990).
7. R. VAN WESEP. Wadge degrees and descriptive set theory. In A. S. KECHRIS AND Y. N. MOSCHOVAKIS, editors, "Cabal Seminar 76–77", vol. 689 of "Lecture Notes in Mathematics", pp. 151–170 (1978). Springer-Verlag.
8. K. W. WAGNER. Eine topologische Charakterisierung einiger Klassen regulärer Folgenmengen. *Elektron. Informationsverarb. Kybernet.* **13**(9), 473–487 (1977).
9. K. W. WAGNER. On ω-regular sets. *Information and Control* **43**(2), 123–177 (1979).
10. H. YOO. Ein effizienter Algorithmus zur Bestimmung des Rabin-Index in Muller-Automaten. Diploma thesis, Inst. f. Inform. u. Prakt. Math, CAU Kiel, Germany (1994). 59 pages.