Computing and Informatics, Vol. 25, 2006, 81–97

MODEL CHECKING OF REGCTL[*]

Tomáš Brázdil, Ivana Černá

Faculty of Informatics, Masaryk University
Botanická 68a
602 00 Brno, Czech Republic
e-mail: {xbrazdil,cerna}@fi.muni.cz

Manuscript received 10 March 2005; revised 1 December 2005
Communicated by Igor Walukiewicz

Abstract. The paper is devoted to the problem of extending the temporal logic CTL so that it is more expressive and complicated properties can be expressed in a more readable form. The specification language RegCTL, an extension of CTL, is proposed. In RegCTL every CTL temporal operator is augmented with a regular expression, thus restricting moments when the validity is required. We propose a local distributed model checking algorithm for RegCTL and exactly state the complexity of model checking RegCTL formulas.

Keywords: Finite state system, regular expression, tree automaton, temporal logic, model checking

1 INTRODUCTION

Model checking is a very successful method for verification of complex reactive systems. A desired behavioural property of a reactive system is specified as a formula of temporal logic, while a formal description of a system is usually transformed into a transition system (Kripke structure). Model checking algorithms verify that the system under study satisfies its expected behavioural specifications.

A key issue in developing model checking algorithms is the choice of a specification language in which a desired behaviour is described. The most common specification languages are temporal logics. Linear time temporal logic formulas are

[*] Research supported by the Grant Agency of Czech Republic, grant No. 201/03/0509

interpreted over linear sequences, while in branching temporal logics each moment in time may split into various possible futures. Among the linear time logics the logic LTL can express precisely the star-free ω-regular behaviours. Nevertheless, there are natural "regular" linear behaviours which cannot be expressed in this logic, as e.g. the behaviour stating that an atomic proposition p is true in all even moments of time. Beside this, the specification of many useful properties is cumbersome for users. To widen its expressibility several extensions were proposed. [13] suggested to use ω-automata as temporal connectives and [7] strengthens the *until* operator of LTL by indexing it with regular programs of propositional dynamic logic. A similar approach to widen expressibility of branching time logics has been advocated in [6] using deterministic ω-automata connectives, in [3], [2] proposing the RCTL logic, and in [10] for the alternation free μ-calculus.

In this paper we concentrate on branching time logics. The contribution of the paper is twofold. Firstly, we generalise CTL logic adopting the approach from [7] and augment the *until* operator with a regular expression. The resulting logic, which subsumes both CTL and RCTL logics, is called RegCTL. Secondly, we elaborate a model checking algorithm for RegCTL logic and exactly state the complexity of model checking problem of RegCTL. We stress that the proposed model checking algorithm is well-distributable and can be implemented within distributed verification tools like `DiVinE Development Library` [1]. In fact, its distribution can be done in the same manner as for the alternation-free μ-calculus. Thus, we have a logic which extends both CTL and the previous extensions of CTL (namely RCTL), but at the same time it admits effective model checking.

RegCTL is in fact a natural extension of CTL. Intuitively, if the system is defined over a set AP of atomic propositions, then an infinite behaviour of the system can be viewed as a word over the alphabet 2^{AP}. A set of allowed behaviours can be described by a regular expression whose alphabet consists of Boolean formulas over AP. In RegCTL, every CTL temporal operator is augmented by a regular expression restricting moments when the validity is required. Both CTL and RCTL temporal operators can be directly formulated in RegCTL.

For model checking RegCTL logic we use an automata theoretic approach presented in [9]. It is based on a translation of RegCTL formula into hesitant/weak symmetric alternating tree automaton. The model checking problem can be then reduced to checking nonemptiness of 1-letter simple weak alternating word automaton. Employing methods from the paper [5] we attain a distributed local model checking algorithm (i.e., it computes the necessary part of a transition system *on-the-fly*).

Contrary to CTL, the size of the automaton corresponding to the formula can be exponential and therefore the model checking of RegCTL is in PSPACE. Nevertheless, we identify a large subset of RegCTL formulas (so called *det*-RegCTL), subsuming e.g. whole RCTL, for which the model checking problem is in P (in fact it is quadratic with respect to the formula size and linear with respect to the size of Kripke structure). We prove that the provided algorithms are optimal in the sense that the model checking problem of RegCTL is PSPACE-hard and the model checking problem for *det*-RegCTL is P-hard.

The paper is organised as follows. We introduce the syntax and semantics of the RegCTL temporal logic in Section 2, and come up with an alternating automaton accepting models of a RegCTL formula in Section 3. Sequential model checking algorithm for this logic is proposed in Section 4. The corresponding distributed model checking procedure is presented in Section 5. Complexity results are provided in Section 6. We give our conclusions in Section 7.

2 THE REGCTL LOGIC

In this section we define the syntax and semantics of Regular CTL (RegCTL) logic, which extends the CTL logic [4] with regular expressions.

Given a finite set X, let $\mathcal{B}(X)$ be the set of all Boolean formulas over X (i.e., Boolean formulas built from elements in X using $\wedge$, $\vee$ and $\neg$), where we also allow the formulas **true** and **false**. If only connectives $\wedge$ and $\vee$ are allowed, we talk about the set of *positive* Boolean formulas over X, $\mathcal{B}^+(X)$. For a set $S \subseteq X$ and a formula $\phi \in \mathcal{B}(X)$, we say that S *satisfies* ϕ, $S \models \phi$, if assigning **true** to elements of S and assigning **false** to elements in $X \setminus S$ makes ϕ true. The length $\|f\|$ of formula $f \in \mathcal{B}(X)$ is defined inductively: $\|\mathbf{true}\| = \|\mathbf{false}\| = \|p\| = 1$ for $p \in X$; $\|g \vee h\| = \|g \wedge h\| = \|g\| + \|h\| + 1$; $\|\neg g\| = \|g\| + 1$.

For a given set $\mathcal{B}(X)$ of Boolean formulas, the set $\mathcal{R}$ of *regular expressions* over $\mathcal{B}(X)$ is the least set containing $\mathcal{B}(X)$ and such that if $P, Q \in \mathcal{R}$ then also $P + Q$, PQ, $P^* \in \mathcal{R}$. Let us denote the language defined by a regular expression R over $\mathcal{B}(X)$ as $\mathcal{L}(R)$ (the alphabet of $\mathcal{L}(R)$ is an appropriate subset of $\mathcal{B}(X)$). The length $\|R\|$ of regular expression R is defined inductively: if $R = f$ for some $f \in \mathcal{B}(X)$, then $\|R\| = \|f\|$; otherwise $\|P + Q\| = \|PQ\| = \|P\| + \|Q\| + 1$; $\|P^*\| = \|P\| + 1$.

2.1 Syntax of RegCTL

Let AP be a set of atomic propositions. An RegCTL *state* formula is either:

- **true**, **false**, p, $\neg p$ for all $p \in AP$,
- $\phi \vee \psi$ or $\phi \wedge \psi$, where ϕ and ψ are RegCTL state formulas,
- $A\phi$ or $E\phi$, where ϕ is a RegCTL path formula.

An RegCTL *path* formula is:

- $\phi U^R \psi$ or $\phi \tilde{U}^R \psi$, where ϕ and ψ are RegCTL state formulas and R is a regular expression over $\mathcal{B}(AP)$ such that $\epsilon \notin \mathcal{L}(R)$.

The *closure* $cl(\tau)$ of a RegCTL formula τ is the set of all RegCTL state subformulas including τ but excluding **true** and **false**. Moreover, we define the multiset $reg\_occ(\tau)$ representing *all occurences* of regular expressions in formula τ. The length $\|\tau\|$ of a RegCTL formula τ is defined as $|cl(\tau)| + \Sigma_{R \in reg\_occ(\tau)} \|R\|$.

2.2 Semantics of RegCTL

The semantics of RegCTL is defined with respect to computation trees. A *tree* is a set $T \subseteq \mathbb{N}^*$ such that if $x.c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$, and for all $0 \leq c' < c$, $x.c' \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x.c$, where $c \in \mathbb{N}$ are the *successors* of x. The number of successors of x is called *degree* of x and is denoted by $d(x)$. The node with no successors is called *leaf*. A *path* π in a tree T is a minimal set $\pi \subseteq T$ containing some node as its root and such that for every $x \in \pi$, either x is a leaf or there exists a unique $c \in \mathbb{N}$ such that $x.c \in \pi$. A tree containing a unique path starting in ϵ is called an (in)finite *word.* Given an alphabet Σ, a Σ*-labeled tree* is a pair $\mathcal{T} = \langle T, L\rangle$ where T is a tree and $L : T \longrightarrow \Sigma$ maps each node of T to a letter in Σ. A *computation tree* is a Σ*-labeled tree* $\mathcal{T}$, where $\Sigma = 2^{AP}$.

The notation $\mathcal{T}, x \models \phi$ indicates that a RegCTL state formula ϕ holds at the node x of the computation tree $\mathcal{T}$. Similarly, $\mathcal{T}, \pi \models \psi$ indicates that a RegCTL path formula ψ holds true along the path π. When $\mathcal{T}$ is clear from the context, we write $x \models \phi$ and $\pi \models \psi$. Also, $\mathcal{T} \models \phi$ if and only if $\mathcal{T}, \epsilon \models \phi$. For a finite sequence of nodes $x_0, x_1, \ldots, x_n$ and a regular expression R over $\mathcal{B}(AP)$ we write $x_0x_1 \ldots x_n \in \mathcal{L}(R)$ iff there exists a word $f_0f_1 \ldots f_n \in \mathcal{L}(R)$ such that $L(x_i) \models f_i$ for all $0 \leq i \leq n$.

The relation $\models$ is inductively defined as follows:

- $x \models \textbf{true}$ and $x \not\models \textbf{false}$
- $x \models p$ for $p \in AP$ iff $p \in L(x)$
- $x \models \neg p$ for $p \in AP$ iff $p \notin L(x)$
- $x \models \phi \vee \psi$ iff $x \models \phi$ or $x \models \psi$
- $x \models \phi \wedge \psi$ iff $x \models \phi$ and $x \models \psi$
- $x \models A\psi$ iff for each path $\pi = \pi_0\pi_1 \cdots$, such that $\pi_0 = x$, we have $\pi \models \psi$
- $x \models E\psi$ iff there exists a path $\pi = \pi_0\pi_1 \cdots$, such that $\pi_0 = x$ and $\pi \models \psi$
- $\pi \models \phi U^R \psi$ iff there exists $i \geq 0$ and $\pi_0\pi_1 \cdots \pi_i \in \mathcal{L}(R)$ such that $\pi_j \models \phi$ for all $0 \leq j < i$ and $\pi_i \models \psi$
- $\pi \models \phi \tilde{U}^R \psi$ iff for all $i \geq 0$ such that $\pi_0\pi_1 \cdots \pi_i \in \mathcal{L}(R)$ the following property holds: if $\pi_i \not\models \psi$, then there exists $0 \leq j < i$ such that $\pi_j \models \phi$.

Usual temporal operators can be expressed as follows: next operator $X\phi$ as $\textbf{true}U^{\textbf{true}\cdot\textbf{true}}\phi$, until operator $\phi U \psi$ as $\phi U^{\textbf{true}\cdot\textbf{true}^*}\psi$, and release operator $\phi \tilde{U} \psi$ as $\phi \tilde{U}^{\textbf{true}\cdot\textbf{true}^*}\psi$.

Let us consider the RegCTL formula $E(q\tilde{U}^{\textbf{true}\cdot(\textbf{true}\cdot\textbf{true})^*}p)$ which expresses the fact that there exists a path where p holds at every even position and this property can be released by q. This property can be expressed neither in CTL nor in RCTL.

The RegCTL formula $A(\mathbf{false}\tilde{U}^{w\cdot b^*\cdot a\cdot(v^*\cdot r+v^*\cdot w\cdot b^*\cdot r)}d)$ (see [3]) illustrates the way how regular expressions can make the formulation of a property easier. The CTL formula expressing the same property is

$$AG(\neg(w\wedge(EX(E[bU(a\wedge(EX(((E[vU(r\wedge\neg d])\wedge$$

$$(E[vU(w\wedge(EX(E[bU(r\wedge\neg d]))))])))))]))))).$$

3 ALTERNATING TREE AUTOMATON FOR REGCTL FORMULA

The model checking algorithm for RegCTL we are going to present is based on a translation of a RegCTL state formula to an automaton over infinite trees which accepts models of the formula (in a similar way as for CTL [9]).

3.1 Alternating tree automata

A *symmetric finite alternating tree automaton* is a tuple $\mathcal{A}=\langle\Sigma,Q,\delta,q^0,F\rangle$, where Σ is an input alphabet, Q is a finite set of states, $\delta: Q\times\Sigma\longrightarrow\mathcal{B}^+(\{\Diamond,\Box\}\times Q)$ is a transition function, q^0 is an initial state. The set F specifies an acceptance condition. We define the *size* $\|\mathcal{A}\|$ of an automaton $\mathcal{A}$ as $|Q|+|F|+\|\delta\|$ where $\|\delta\|$ is the sum of the lengths of the non-identically false formulas that appear as $\delta(q,\sigma)$ for some $q\in Q$ and $\sigma\in\Sigma$.

A *run* $\langle T_r,r\rangle$ of an alternating automaton $\mathcal{A}$ over a Σ-labelled tree $\langle T,L\rangle$ is a Σ_r-labelled tree where $\Sigma_r=\mathbb{N}^*\times Q$ and $\langle T_r,r\rangle$ satisfies the following:

- $r(\epsilon)=(\epsilon,q_0)$,
- Let $y\in T_r$ with $r(y)=(x,q)$ and $\delta(q,L(x))=\Theta$. Then there is a (possibly empty) set $S=\{(c_0,q_0),(c_1,q_1),\ldots,(c_n,q_n)\}\subseteq\{0,\ldots,d(x)-1\}\times Q$ such that the following holds:
  1. S satisfies Θ, where $(\Diamond,p)\Leftrightarrow(0,p)\vee\ldots\vee(d(x)-1,p)$ and $(\Box,p)\Leftrightarrow(0,p)\wedge\ldots\wedge(d(x)-1,p)$ for $p\in Q$,
  2. for all $0\le i\le n$, we have $y.i\in T_r$ and $r(y.i)=(x.c_i,q_i)$.

We consider an alternating *word* automaton to be a special case of a tree automaton with transition function $\delta: Q\times\Sigma\longrightarrow\mathcal{B}^+(Q)$.

Given a run $\langle T_r,r\rangle$ and an infinite path π in T_r, let $inf(\pi)\subseteq Q$ be such that $q\in inf(\pi)$ iff there are infinitely many $y\in\pi$ for which $r(y)\in\mathbb{N}^*\times\{q\}$ (i.e., $inf(\pi)$ is the set of states which appear infinitely often in π). A run $\langle T_r,r\rangle$ is accepting iff all of its infinite paths satisfy the acceptance condition. We denote $\mathcal{L}(\mathcal{A})$ the set of all computation trees for which there is an accepting run of $\mathcal{A}$.

Here we consider two special types of alternating tree automata, so called *hesitant* automata (HAA) and *weak* automata (WAA), with special restrictions on the transition function and specific acceptance conditions.

In a hesitant automaton there exists a partition of Q into disjoint sets $Q_1, \ldots, Q_m$ and a partial order $\leq$ on the collection of Q_i's such that for each $q \in Q_i$ and $q' \in Q_j$ for which q' occurs in $\delta(q, \sigma)$ we have $Q_j \leq Q_i$. In addition, each set Q_i is classified as either *transient*, *existential* or *universal*. The type of Q_i is determined by the following rules:

- Q_i is a *transient* set iff for all $q \in Q_i$ and $\sigma \in \Sigma$, $\delta(q, \sigma)$ contains no element with a state from Q_i.
- Q_i is an *existential* set iff for all $q \in Q_i$ and $\sigma \in \Sigma$, $\delta(q, \sigma)$ contains only disjunctively related elements of the form $(\Diamond, p)$ where $p \in Q_i$.
- Q_i is an *universal* set iff for all $q \in Q_i$ and $\sigma \in \Sigma$, $\delta(q, \sigma)$ contains only conjunctively related elements of the form $(\Box, p)$ where $p \in Q_i$.

The acceptance condition is a tuple $\langle G, B \rangle$, where $G, B \subseteq Q$. Every infinite path π in T_r gets trapped within some existential or universal set Q_i. The path π then satisfies an acceptance condition $\langle G, B \rangle$ iff

- either Q_i is an existential set and $\mathit{inf}(\pi) \cap G \neq \emptyset$
- or Q_i is an universal set and $\mathit{inf}(\pi) \cap B = \emptyset$.

The *depth* of HAA is defined as a maximal length of a chain in the partial order $\leq$ on the collection of Q_i's.

In a weak automaton there exists a partition of Q into $Q_1, \ldots, Q_m$ with the same partial order as in HAA. The acceptance condition F is a subset of Q such that for every Q_i, $1 \leq i \leq m$, either $Q_i \subseteq F$ (Q_i is an accepting set) or $Q_i \cap F = \emptyset$ (Q_i is a rejecting set).

Next, we show how to complement hesitant and weak automata. For two alternating automata $\mathcal{A}_1$ and $\mathcal{A}_2$ over the same alphabet Σ we say that $\mathcal{A}_2$ complements $\mathcal{A}_1$ iff $\mathcal{L}(\mathcal{A}_2)$ includes exactly all the Σ-labelled trees that are not in $\mathcal{L}(\mathcal{A}_1)$.

Given a transition function δ, let $\tilde{\delta}$ denote the dual function of δ. That is, for every q and σ with $\delta(q, \sigma) = \theta$, let $\tilde{\delta}(q, \sigma) = \tilde{\theta}$, where $\tilde{\theta}$ is obtained from θ by switching $\vee$ and $\wedge$, by switching $(\Diamond, p)$ and $(\Box, p)$ for each p and by switching **true** and **false**. For example, if $\theta = ((\Box, p) \wedge (\Diamond, q)) \vee \mathbf{true}$ then $\tilde{\theta} = ((\Diamond, p) \vee (\Box, q)) \wedge \mathbf{false}$.

Theorem 1.

- Given a hesitant alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, \langle G, B \rangle \rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_0, \langle B, G \rangle \rangle$ is a hesitant automaton that complements $\mathcal{A}$.
- Given a weak alternating automaton $\mathcal{A} = \langle \Sigma, Q, \delta, q_0, F \rangle$, the alternating automaton $\tilde{\mathcal{A}} = \langle \Sigma, Q, \tilde{\delta}, q_0, Q \setminus F \rangle$ is a weak automaton that complements $\mathcal{A}$.

Proof. For hesitant automata is the assertion proved in [9].

For weak automaton $\mathcal{A}$ one can easily see that $\tilde{\mathcal{A}}$ is weak as the partition of Q into sets and the partial order over them hold also with respect to $\tilde{\mathcal{A}}$. According to [11], in order to prove that $\tilde{\mathcal{A}}$ complements $\mathcal{A}$ it is enough to prove that for every path $\pi \in Q^\omega$, we have that π satisfies the acceptance condition of $\mathcal{A}$ in a run of $\mathcal{A}$ iff π does not satisfy the acceptance condition of $\tilde{\mathcal{A}}$ in a run of $\tilde{\mathcal{A}}$. However, for weak automata this can be verified easily. $\square$

Remark 1. We say that $\tilde{\mathcal{A}}$ is the *dual* of $\mathcal{A}$.

3.2 Translation of a RegCTL formula to an automaton

Let us first fix some notation. For an RegCTL formula τ the multiset $reg\_occ(\tau) = \{R_1, \dots, R_n\}$ represents all occurences of regular expressions in the formula. For every regular expression R_i we have a finite state automaton $\mathcal{A}_i = (Q_i, \Sigma_i, \delta_i, q_i^0, F_i)$, $\Sigma_i \subseteq \mathcal{B}(AP)$, which accepts exactly $\mathcal{L}(R_i)$. We suppose all Q_i's to be pairwise disjunctive. For states of these automata we use symbols r, q (with indices, if necessary). Moreover, let for $r \in Q_i$ and $\sigma \in 2^{AP}$ the symbol $succ(r, \sigma)$ denote the set of states $\bigcup_{\sigma \models f} \delta_i(r, f)$.

Given a RegCTL formula τ we construct the weak symmetric alternating automaton $\mathcal{A}_\tau = (2^{AP}, Q, \delta, \tau, F)$. The set of states of the automaton $\mathcal{A}_\tau$ is $Q = (\bigcup_{i=1,\dots n} Q_i) \cup cl(\tau)$. For all $\sigma \in 2^{AP}$ its transition function δ is defined inductively as follows:

1. $\delta(p, \sigma) = \textbf{true}$ if $p \in \sigma$ and $\delta(p, \sigma) = \textbf{false}$ if $p \notin \sigma$.
2. $\delta(\neg p, \sigma) = \textbf{true}$ if $p \notin \sigma$ and $\delta(\neg p, \sigma) = \textbf{false}$ if $p \in \sigma$.
3. $\delta(\phi \vee \psi, \sigma) = \delta(\phi, \sigma) \vee \delta(\psi, \sigma)$.
4. $\delta(\phi \wedge \psi, \sigma) = \delta(\phi, \sigma) \wedge \delta(\psi, \sigma)$.
5. $\delta(E(\phi U^{R_i} \psi), \sigma) = \delta(q_i^0, \sigma)$ and for $r \in Q_i$
$$\delta(r,\sigma) = \begin{cases} \bigvee_{q \in succ(r,\sigma)} (\Diamond, q) \wedge \delta(\phi, \sigma) & \text{if } succ(r,\sigma) \cap F_i = \emptyset \\ (\bigvee_{q \in succ(r,\sigma)} (\Diamond, q) \wedge \delta(\phi, \sigma)) \vee \delta(\psi, \sigma) & \text{otherwise.} \end{cases}$$
6. $\delta(A(\phi \tilde{U}^{R_i} \psi), \sigma) = \delta(q_i^0, \sigma)$ and for $r \in Q_i$
$$\delta(r,\sigma) = \begin{cases} \bigwedge_{q \in succ(r,\sigma)} (\Box, q) \vee \delta(\phi, \sigma) & \text{if } succ(r,\sigma) \cap F_i = \emptyset \\ (\bigwedge_{q \in succ(r,\sigma)} (\Box, q) \vee \delta(\phi, \sigma)) \wedge \delta(\psi, \sigma) & \text{otherwise.} \end{cases}$$
7. $\delta(E(\phi \tilde{U}^{R_i} \psi), \sigma) = \delta(q_i^0, \sigma)$ and for $r \in Q_i$
$$\delta(r,\sigma) = \begin{cases} \bigwedge_{q \in succ(r,\sigma)} (\Diamond, q) \vee \delta(\phi, \sigma) & \text{if } succ(r,\sigma) \cap F_i = \emptyset \\ (\bigwedge_{q \in succ(r,\sigma)} (\Diamond, q) \vee \delta(\phi, \sigma)) \wedge \delta(\psi, \sigma) & \text{otherwise.} \end{cases}$$
8. $\delta(A(\phi U^{R_i} \psi), \sigma) = \delta(q_i^0, \sigma)$ and for $r \in Q_i$
$$\delta(r,\sigma) = \begin{cases} \bigvee_{q \in succ(r,\sigma)} (\Box, q) \wedge \delta(\phi, \sigma) & \text{if } succ(r,\sigma) \cap F_i = \emptyset \\ (\bigvee_{q \in succ(r,\sigma)} (\Box, q) \wedge \delta(\phi, \sigma)) \vee \delta(\psi, \sigma) & \text{otherwise.} \end{cases}$$

Remark 2. We define an empty disjunction to be false and empty conjunction to be true.

The automaton $\mathcal{A}_\tau$ is weak. The acceptance condition is $F = \bigcup Q_i$ for all Q_i's such that the regular expression R_i occurs in a subformula of the form $E(\phi \tilde{U}^{R_i} \psi)$ or $A(\phi \tilde{U}^{R_i} \psi)$. The weakness partition over the set of states is formed by singletons $\{\psi\}$, $\psi \in cl(\tau)$, and by all sets Q_i, $1 \leq i \leq n$.

The correctness of the given construction is guaranteed only for cases where for every regular expression R_i which occurs in a subformula of the form $E(\phi \tilde{U}^{R_i} \psi)$ or $A(\phi U^{R_i} \psi)$ the corresponding finite automaton $\mathcal{A}_i$ is *deterministic.*

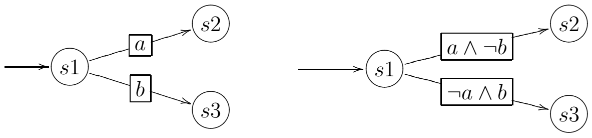

Fig. 1. The automaton on the left hand side is non-deterministic while the automaton on the right hand side is deterministic

Here deterministic automaton is an automaton such that for its arbitrary state $r \in Q_i$ and $\sigma \in 2^{AP}$ the cardinality of the set $succ(r, \sigma)$ is at most one (in the succeeding text we always use this notion of determinism). For example, the automaton on the left hand side of Figure 1 is not deterministic as $succ(s_1, \{a, b\}) = \{s_2, s_3\}$. To explain problems caused by nondeterministic automata let us consider the formula $\tau \equiv E(\mathbf{false} \tilde{U}^R g)$, with R specified in Figure 2, and the computation tree from Figure 3.

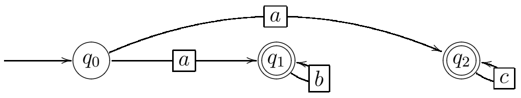

Fig. 2. Finite state automaton $\mathcal{A}$ for the regular expression R; edges are labelled with atomic propositions (i.e. formulas over AP); q_1 and q_2 are accepting, q_0 is initial

$$\begin{tikzcd}
 & & \{b\} \arrow[r] & \{b\} \arrow[r] & \cdots \\
\{a,g\} \arrow[r] & \{b,c,g\} \arrow[ur] \arrow[dr] & & & \\
 & & \{c\} \arrow[r] & \{c\} \arrow[r] & \cdots
\end{tikzcd}$$

Fig. 3. Computation tree; nodes are labelled with atomic propositions true in them

The automaton $\mathcal{A}_\tau$ in state $E(\mathbf{false} \tilde{U}^R g)$ ($\equiv q_0$) reading $\{a, g\}$ proceeds conjunctively to states q_1 and q_2 and to the node labelled $\{b, c, g\}$. Being in state q_1 and reading $\{b, c, g\}$, $\mathcal{A}_\tau$ remains in q_1 and disjunctively proceeds to the node labelled $\{c\}$. Being in state q_2 and reading $\{b, c, g\}$, $\mathcal{A}_\tau$ remains in q_2 and disjunctively proceeds to the node labelled $\{b\}$. Both paths are finite and accepting and thus $\mathcal{A}_\tau$ accepts, but τ is not true in the node labelled $\{a, g\}$.

Remark 3. Assuming the determinism of relevant finite automata, the automaton $\mathcal{A}_\tau$ is also hesitant. The hesitant partition is the same as the weakness partition. The set Q_i is existential iff the regular expression R_i occurs in a subformula of the form $E(\phi U^{R_i} \psi)$ or $E(\phi \tilde{U}^{R_i} \psi)$. The set Q_i is universal iff the regular expression R_i occurs in a subformula of the form $A(\phi U^{R_i} \psi)$ or $A(\phi \tilde{U}^{R_i} \psi)$. Other sets are transient. The acceptance condition for automaton $\mathcal{A}_\tau$ is $F = \langle G, B \rangle$ where

- $G = \bigcup Q_i$ for all Q_i such that the regular expression R_i occurs in a subformula of the form $E(\phi \tilde{U}^{R_i} \psi)$ and
- $B = \bigcup Q_i$ for all Q_i such that the regular expression R_i occurs in a subformula of the form $A(\phi U^{R_i} \psi)$.

In what follows we suppose that finite automata for regular expressions R_i which occur in subformulas of the form $E(\phi \tilde{U}^{R_i} \psi)$ or $A(\phi U^{R_i} \psi)$ are deterministic.

Theorem 2. Let $\mathcal{T} = \langle T, L \rangle$ be a computation tree. Then the automaton $\mathcal{A}_\tau$ accepts $\mathcal{T}$ if and only if $\mathcal{T} \models \tau$.

Proof. We first prove that $\mathcal{A}_\tau$ is complete. That is, given a computation tree $\mathcal{T}$, a formula $\varphi \in cl(\tau)$ and a node x for which $\mathcal{T}, x \models \varphi$, then $\mathcal{A}_\tau$ accepts the subtree of the computation tree $\mathcal{T}$ with root x starting in the state φ. Thus, in particular, if $\mathcal{T} \models \tau$ then $\mathcal{A}_\tau$ accepts $\mathcal{T}$.

To this end we use the following notation. For the finite automaton $\mathcal{A}_i$, its states r, q and a node $x \in T$ we use $q \in \delta_i(r, x)$ as an abbreviation for $q \in \delta_i(r, f)$ where $f \in \mathcal{B}(AP)$ and $L(x) \models f$. A computation of $\mathcal{A}_i$ over $x_0 \cdots x_n$ is a sequence of states $q_0, \dots, q_n$ such that q_0 is an initial state and $q_{j+1} \in \delta_i(q_j, x_j)$ for $0 \leq j < n$. If, moreover, the condition $\delta_i(q_n, x_n) \cap F_i \neq \emptyset$ is true, then the computation is *accepting.* We prove the completeness by induction on the structure of φ. Cases $\varphi = p$, $\varphi = \neg p$, $\varphi = \phi \vee \psi$, $\varphi = \phi \wedge \psi$ are simple.

- $x_0 \models E(\phi U^{R_i} \psi)$
  There is a path $x_0 \cdots x_n$ in T such that $x_0 \cdots x_n \in \mathcal{L}(R_i)$, $x_j \models \phi$ for $0 \leq j < n$ and $x_n \models \psi$. Let $q_0, \dots, q_n$ be an accepting computation of $\mathcal{A}_i$ over $x_0 \cdots x_n$.

  $\mathcal{A}_\tau$ disjunctively chooses states q_j and input nodes x_j. In every node x_j automaton $\mathcal{A}_\tau$ conjunctively proceeds as if it were in the state ϕ. Because $\delta_i(q_n, x_n) \cap F_i \neq \emptyset$, $\mathcal{A}_\tau$ in the state q_n proceeds as if it were in the state ψ.

- $x_0 \models E(\phi \tilde{U}^{R_i} \psi)$
  There is a path $x_0 \cdots$ in T such that for each of its prefixes $x_0 \cdots x_n$, the following holds: if $x_0 \cdots x_n \in \mathcal{L}(R_i)$ then either $x_n \models \psi$ or there exists $0 \leq k < n$ such that $x_k \models \phi$.

  $\mathcal{A}_\tau$ reading a node x_j and being in a state q proceeds as follows: if $x_j \models \phi$ then it proceeds as if it were in the state ϕ. If $\delta_i(q, x_j) = \emptyset$ then $\mathcal{A}_\tau$ does not continue along this path. Otherwise it proceeds to x_{j+1} and to the only successor state of q according to δ_i. If $\delta_i(q, x_j) \cap F_i \neq \emptyset$ then $x_0 \cdots x_j \in \mathcal{L}(R_i)$ and $x_k \not\models \phi$ for $0 \leq k < j$ and $x_j \models \psi$, thus the automaton conjunctively proceeds as if it were in the state ψ. If ϕ does not hold along $x_0 \cdots$, then the path is accepting due to the acceptance condition.

We now prove that $\mathcal{A}_\tau$ is sound. That is, given an accepting run $\langle T_r, r\rangle$ of $\mathcal{A}_\tau$ over a computation tree $\mathcal{T} = \langle T, L\rangle$, we prove that for every $y \in T_r$ such that $r(y) = (x, \varphi)$, $\varphi \in cl(\tau)$, we have $\mathcal{T}, x \models \varphi$. Thus, in particular, $\mathcal{T}, \epsilon \models \tau$.

Let $\langle T_r, r\rangle$ be a run of the alternating automaton $\mathcal{A}_\tau$ over a computation tree $\mathcal{T} = \langle T, L\rangle$. We describe a path in the run T_r as a sequence of its labels (i.e., a sequence of tuples (x, q) where $x \in T$ and $q \in Q$). Let $(x_0, q_0) \cdots$ be a finite or infinite path in T_r. We say that its prefix *pr* is *maximal* in Q' iff either $pr = (x_0, q_0) \cdots (x_n, q_n)$, $q_0, \ldots, q_n \in Q'$ and $q \notin Q'$ for every successor (q, x) of (q_n, x_n) in T_r, or $pr = (x_0, q_0) \cdots$ is infinite and $q_j \in Q'$ for $0 \leq j$. The projection of a path $\pi = (x_0, q_0) \cdots$ is $proj(\pi) = x_0 \cdots$.

The proof proceeds by induction on the structure of φ. Cases $\varphi = p$, $\varphi = \neg p$, $\varphi = \phi \vee \psi$, $\varphi = \phi \wedge \psi$ are simple. In the next construction we make use of the fact that in $\mathcal{A}_\tau$ we have several names for one state.

- $r(y) = (x_0, E(\phi U^{R_i} \psi))$
  Let *pr* be a prefix maximal in Q_i of a path in T_r starting with $(x_0, E(\phi U^{R_i} \psi))$. Due to the acceptance conditions the prefix $pr = (x_0, E(\phi U^{R_i} \psi)) \cdots (x_n, q_n)$ is finite. Then $\mathcal{A}_\tau$ in the state q_n reading x_n must proceed as if it were in the state ψ assuring $x_n \models \psi$ and $x_0 \cdots x_n \in \mathcal{L}(\mathcal{R}_i)$. Moreover, along this prefix it must conjunctively proceed as if it were in the state ϕ assuring $x_j \models \phi$ for $0 \leq j < n$.
- $r(y) = (x_0, E(\phi \tilde{U}^{R_i} \psi))$
  $\mathcal{A}_\tau$ disjunctively chooses a path in T following states of the only possible computation of $\mathcal{A}_i$. Let *pr* be a prefix maximal in Q_i of a path in T_r starting with $(x_0, E(\phi \tilde{U}^{R_i} \psi))$.

  Case 1: If *pr* is infinite (it is possible due to the acceptance condition) then thanks to the definition of δ and determinism of $\mathcal{A}_i$ we have that whenever a prefix of $proj(pr)$ is in $\mathcal{L}(\mathcal{R}_i)$ then $\mathcal{A}_\tau$ proceeds as if it were in the state ψ.

  Case 2: Otherwise $pr = (x_0, E(\phi \tilde{U}^{R_i} \psi)) \cdots (x_n, q_n)$. If $\delta_i(q_n, x_n) = \emptyset$, then no word with the prefix $x_0 \cdots x_n$ is in $\mathcal{L}(\mathcal{R}_i)$. If $\delta_i(q_n, x_n) \neq \emptyset$, then $\mathcal{A}_\tau$ proceeds in the state q_n reading x_n as if it were in the state ϕ, assuring thus $x_n \models \phi$. Note that whenever a prefix of $x_0 \cdots x_n$ is in $\mathcal{L}(\mathcal{R}_i)$ then $\mathcal{A}_\tau$ proceeds as if it were in the state ψ (similar arguments as in Case 1).

Formulas of the form $A(\phi \tilde{U}^{R_i} \psi))$ are treated in the following manner: formula $A(\phi \tilde{U}^{R_i} \psi)$ is dual to the formula $E(\phi U^{R_i} \psi)$, that is, $A(\phi \tilde{U}^{R_i} \psi) \equiv \neg E(\neg \phi U^{R_i} \neg \psi)$. Therefore the dual of an automaton for $E(\neg \phi U^{R_i} \neg \psi)$ is an automaton for $A(\phi \tilde{U}^{R_i} \psi)$.

However, this is *the* automaton for $A(\phi\tilde{U}^{R_i}\psi)$ as we have defined it in the construction.

Formula $A(\phi U^{R_i}\psi)$ is dual to the formula $E(\phi\tilde{U}^{R_i}\psi)$ and thus analogical arguments prove the correctness for formulas of the form $A(\phi U^{R_i}\psi)$. □

4 SEQUENTIAL REGCTL MODEL CHECKING

At first we define the Kripke structure as a tuple $K = \langle AP, W, E, w^0, L\rangle$ where AP is a set of atomic propositions as defined above, W is a set of states, $E \subseteq W \times W$ is a transition relation that must be total (i.e., for every $w \in W$ there exists $w' \in W$ such that $\langle w, w'\rangle \in E$), w^0 is an initial state, and $L : W \to 2^{AP}$ maps each state to the set of atomic propositions true in that state.

We define the size $\|K\|$ of K as $|W| + |E|$. Every Kripke structure $K = \langle AP, W, E, w^0, L\rangle$ can be viewed as a 2^{AP}-labelled computation tree $\mathcal{T}_K = \langle T_K, L_K\rangle$ obtained by unwinding K.

The model checking problem is for given temporal logic formula τ and Kripke structure K to decide whether $\mathcal{T}_K \models \tau$. The model checking algorithm for a given RegCTL state formula τ and a Kripke structure K proceeds as follows:

1. construct the alternating automaton $\mathcal{A}_\tau$ as defined above,
2. construct the product automaton $\mathcal{A}_{K,\tau} = K \times \mathcal{A}_\tau$ whose language is nonempty iff $\mathcal{T}_K \models \tau$,
3. check nonemptiness of the product automaton $\mathcal{A}_{K,\tau}$.

The product automaton $\mathcal{A}_{K,\tau}$ is defined as follows: Let $\mathcal{A}_\tau = \langle 2^{AP}, Q_\tau, \delta_\tau, q_0, F_\tau\rangle$ and $K = \langle AP, W, E, w^0, L\rangle$. The product of $\mathcal{A}_\tau$ and K is a 1-letter alternating word automaton $\mathcal{A}_{K,\tau} = \langle\{a\}, W \times Q_\tau, \delta, \langle w^0, q_0\rangle, F\rangle$ where δ and F are defined as follows:

- Let $q \in Q_\tau$, $w \in W$, $succ(w) = \langle w_0, \ldots, w_{d(w)-1}\rangle$ and $\delta_\tau(q, L(w)) = \Theta$. Then $\delta(\langle w, q\rangle, a) = \Theta'$, where Θ' is obtained from Θ by replacing each $(\Diamond, p)$ by $\langle w_0, p\rangle \vee \ldots \vee \langle w_{d(w)-1}, p\rangle$ and each $(\Box, p)$ by $\langle w_0, p\rangle \wedge \ldots \wedge \langle w_{d(w)-1}, p\rangle$.
- The acceptance condition F respects the acceptance condition F_τ of $\mathcal{A}_\tau$. If $\mathcal{A}_\tau$ is weak then $F = W \times F_\tau$. If $\mathcal{A}_\tau$ is hesitant and $F_\tau = \langle G, B\rangle$ then $F = \langle W \times G, W \times B\rangle$.

The product automaton is hesitant (weak) if $\mathcal{A}_\tau$ is hesitant (weak).

Theorem 3. [9] $\mathcal{A}_{K,\tau}$ accepts a^ω iff $\mathcal{T}_K \models \tau$.

Algorithms for nonemptiness check of weak and hesitant automata are given in [9], their complexity is discussed in Section 6.

5 DISTRIBUTED REGCTL MODEL CHECKING

The distributed algorithm is based on a characterisation of the model checking problem in terms of two-person games due to Stirling [12]. This approach has been used in [5] for model checking of alternation-free μ-calculus and formulated as colouring of game graphs. As noted in [5], the procedure can also be understood as a parallel procedure for checking the emptiness of 1-letter simple weak alternating word automata.

The product automaton $\mathcal{A}_{K,\tau}$ we have constructed in Section 3 is a 1-letter weak alternating word automaton. We propose an algorithm for translating it into a simple automaton. Our algorithm is a modification of the one from [9] and is more appropriate for the use in the distributed on-the-fly setting. Consequently we can apply the local distributed model checking algorithms from [5] to the logic RegCTL.

Definition 1. A formula in $\mathcal{B}^+(X)$ is *simple* if it is either atomic or has the form $x * y$, where $* \in \{\wedge, \vee\}$ and $x, y \in X$. An alternating automaton is simple if all its transitions are simple.

Let $\mathcal{A}_{K,\tau} = \langle \{a\}, W \times Q_\tau, \delta, \langle w^0, q_0\rangle, W \times F_\tau\rangle$ be the weak product automaton from Section 4. Let $W\times Q_1, \ldots, W\times Q_m$ be the weak partition of its states such that $W \times Q_1 \leq \ldots \leq W \times Q_m$ is an extension of the partial order to a total order. Our aim is to translate $\mathcal{A}_{K,\tau}$ to a simple automaton $\mathcal{A}^s_{K,\tau} = \langle \{a\}, Q^s, \delta^s, \langle w^0, q_0\rangle, F^s\rangle$. We define Q^s inductively as follows:

- for every $q \in W \times Q_\tau$, we have $q \in Q^s$
- for every $q \in W \times Q_\tau$ with $\delta(q, a) = \theta_1 * \theta_2$, we have $\theta_1, \theta_2 \in Q^s$
- for every $\theta_1 * \theta_2 \in Q^s$, we have $\theta_1, \theta_2 \in Q^s$.

Thus a state in Q^s is either $q \in W \times Q_\tau$ or a strict subformula of a transition in δ. The transition function δ^s is

- $\delta^s(q, a) = \delta(q, a)$ for $q \in W \times Q_\tau$
- $\delta^s(\theta_1 * \theta_2, a) = \theta_1 * \theta_2$.

We claim that the new automaton is weak as well. The partition of Q^s into $Q^s_1, \ldots, Q^s_m$ is as follows. A state $q \in Q^s$ is in Q^s_i iff either $q \in W \times Q_i$ or $q = \theta$ and $i = \max\{j \mid r \text{ occurs in } \theta \text{ and } r \in W \times Q_j\}$. The new acceptance condition is $F^s = \bigcup Q^s_i$ where $W \times Q_i \subseteq W \times F_\tau$. The weakness of $\mathcal{A}^s_{K,\tau}$ can be easily seen from the definition of the partition. The fact $\mathcal{L}(\mathcal{A}_{K,\tau}) \neq \emptyset$ iff $\mathcal{L}(\mathcal{A}^s_{K,\tau}) \neq \emptyset$ can be argumented in the same way as in [9].

We note that the simple version of the product automaton can be computed on-the-fly from the formula and the Kripke structure. The size of the simple product automaton is asymptotically the same as the size of the original one. The important fact is that the partition of the states of the simple automaton can be computed on-the-fly as well using only the knowledge of the partition of $\mathcal{A}_\tau$.

All in all, we have transformed the model checking problem of RegCTL into the emptiness problem of 1-letter simple weak alternating word automata. These automata are in a straightforward manner (as noted in [5]) related to games and therefore we can use distributed algorithms from [5] for checking the emptiness of this kind of automata.

6 COMPLEXITY RESULTS

The complexity of the model checking algorithm depends on the type of the formula. As we have shown in Section 3, the necessary condition for $\mathcal{A}_\tau$ to be correct is the determinism of finite automata for regular expressions occuring in subformulas of the form $E(\phi\tilde{U}^R\psi)$ and $A(\phi U^R\psi)$. For this reason we define a *deterministic fragment* of RegCTL, called *det*-RegCTL. In this fragment R occuring in $A(\phi U^R\psi)$ or $E(\phi\tilde{U}^R\psi)$ are restricted to regular expressions which have deterministic finite automata with the number of states linear with respect to the size of R. For a *det*-RegCTL formula τ it is guaranteed that the number of states of $\mathcal{A}_\tau$ is linear in $\|\tau\|$. For a general RegCTL formula the number of states can be $2^{O(\|\tau\|)}$ due to the necessary determinization. In both cases the length of $\delta_\tau(q,\sigma)$ (for fixed q and σ) is linear in $\|\tau\|$.

The complexity of the model checking algorithm problem is measured with respect to the size of K and τ. The key point is the size of the product automaton.

The number of states of $\mathcal{A}_{K,\tau}$ is $|W|\cdot|Q_\tau|$ and the size of F is $O(|W|\cdot|Q_\tau|)$. The length of $\delta(\langle w,q\rangle,a)$ is equal to the length of $\delta_\tau(q,L(w))$ times the degree of w. Summing up lengths of $\delta((w,q),a)$ for fixed q and all states $w\in W$ gives us $O(|E|\cdot\|\tau\|)$. The total size of the transition function is $O(|E|\cdot\|\tau\|\cdot|Q_\tau|)$. Thus the total size of the product automaton $\mathcal{A}_{K,\tau}$ is $O(\|K\|\cdot\|\tau\|^2)$ for a *det*-RegCTL formula τ and $O(\|K\|\cdot 2^{O(\|\tau\|)})$ for a general RegCTL formula. The depth of $\mathcal{A}_{K,\tau}$ is $O(\|\tau\|)$. We note that $\mathcal{A}_{K,\tau}$ can be computed on-the-fly in time linear with respect to its size.

Theorem 4. [9] The 1-letter nonemptiness problem for hesitant alternating word automata is decidable in linear running time.

Theorem 5. [9] The 1-letter nonemptiness problem for hesitant alternating word automata of size n and depth m is decidable in space $O(m.log^2n)$.

By Theorem 4 we have that the model checking problem for *det*-RegCTL is in P (in fact it can be done in time $O(\|K\|.\|\tau\|^2)$. As the model checking of CTL is P-complete we have that model checking of *det*-RegCTL is P-complete too.

According to Theorem 5 the complexity of the model checking problem for RegCTL is in PSPACE. In what follows we prove that the provided algorithm is optimal in the sense that the problem is PSPACE-hard.

Theorem 6. The RegCTL model checking problem is PSPACE-hard.

Proof. We prove the PSPACE-hardness by reduction from the membership problem for polynomialy space-bounded Turing machines, which is known to be PSPACE-hard.

Let $T=(S,\gamma,\delta,\mathit{Start})$ be a deterministic Turing machine with space complexity n^c (c is a fixed constant). Wlog. we can suppose that T has two special states *Accept, Reject* $\in S$ such that every computation of T eventually gets trapped either in the state *Accept* or in *Reject* and keeps running forever in this state. Moreover, we suppose that the transition function of T is complete. Let us denote w the input of T. Our aim is to construct a Kripke structure K and an RegCTL state formula τ such that T accepts w if and only if K is a model of τ.

Every configuration of the computation of T on an input word w of length n can be written down as a string of length $m = n^c + 1$ over the alphabet $\Gamma = \gamma \cup S$. The computation of T on w is a sequence of configurations. Let us denote the computation σ as $\sigma = \sigma_1, \sigma_2 \ldots$, where $\sigma_i \in \Gamma$ for $i \geq 1$. Now we can define a function f capturing the dependences between symbols in σ. Namely, for each $i \geq 1$, $f(\sigma_i, \sigma_{i+1}, \sigma_{i+2}, \sigma_{i+3}) = \sigma_{i+m+1}$ is the symbol in the subsequent configuration, whose value is completely determined by symbols $\sigma_i, \ldots, \sigma_{i+3}$. For $a, b, c, d \in \gamma$ and $s \in S$ the function defined as follows:

$$
\begin{aligned}
&f(a,b,c,d) = b \\
&f(s,a,b,c) = \begin{cases} a' & \text{if } \delta(s,a) = (s',a',L) \\ s' & \text{if } \delta(s,a) = (s',a',R) \end{cases} \\
&f(a,s,b,c) = \begin{cases} a & \text{if } \delta(s,b) = (s',b',L) \\ b' & \text{if } \delta(s,b) = (s',b',R) \end{cases} \\
&f(a,b,s,c) = \begin{cases} s' & \text{if } \delta(s,c) = (s',c',L) \\ b & \text{if } \delta(s,a) = (s',c',R) \end{cases} \\
&f(a,b,c,s) = b
\end{aligned}
$$

For remaining quadruples the values can be defined arbitrarily. Now we are prepared to define a regular expression R describing all words over the alphabet Γ, which *do not* code an accepting computation of T on $w = w_1 w_2 \cdots w_n$ (string $\sigma_1 \sigma_2 \cdots \sigma_m$ is the code of the initial configuration) :

$$
\begin{aligned}
R &= \neg Correct + \neg Init + \Gamma^* Reject \\
\neg Correct &= \sum_{a,b,c,d \in \Gamma} \Gamma^* a\ b\ c\ d\ \Gamma^{m-3}(\Gamma \setminus f(a,b,c,d)) \\
\neg Init &= (\Gamma \setminus \{\sigma_1\}) + \sigma_1(\Gamma \setminus \{\sigma_2\}) + \sigma_1\sigma_2(\Gamma \setminus \{\sigma_3\}) + \ldots \\
&\quad + \sigma_1\sigma_2 \cdots \sigma_{m-1}(\Gamma \setminus \{\sigma_m\})
\end{aligned}
$$

The RegCTL formula we are looking for is $\tau = E(\mathbf{false}\tilde{U}^R\mathbf{false})$. It remains to define the Kripke structure. We do it in such a way that (labels of) infinite paths in K correspond exactly to words in Γ^ω starting with the initial state *Start* of the Turing machine T. Let $K = \langle \Gamma, W, E, s_1, L \rangle$ where $W = \{s_1, \ldots, s_{|\Gamma|}\}$, $E = W \times W$ and for $\Gamma = \{\gamma_1, \ldots, \gamma_{|\Gamma|}\}$, $\gamma_1 = Start$, we define $L(s_i) = \{\gamma_i\}$ for $1 \leq i \leq |\Gamma|$.

The correctness of the construction is given by the following arguments. Suppose that T accepts w. Then there is an accepting computation of T on w. Let us consider an infinite path π in K which corresponds exactly to this accepting computation. Then no prefix of π is in $\mathcal{L}(R)$ and $K \models \tau$.

On the contrary, suppose that T does not accept w. Then each path π in K corresponds either to an incorrect computation or to a rejecting computation. In the first case there is a prefix u of π such that the last letter of u is the first symbol violating the property "to be a computation". Then u belongs to $\mathcal{L}(\neg Init + \neg Correct)$. In the latter case let u be a prefix of π ending with the symbol *Reject*. Then u belongs to $\mathcal{L}(\Gamma^* Reject)$. In both cases $K \not\models \tau$. □

Corollary 1. Let R be a regular expression over the alphabet $\{0,1\}$. The problem whether each infinite word $u \in \{0,1\}^\omega$ has a finite prefix $v \in \mathcal{L}(R)$ is PSPACE-hard.

Proof. Follows immediately from the proof of Theorem 6 as symbols from Γ can be encoded binary and from the fact that PSPACE is closed under complementation. □

The other way how the complexity of the model checking problem can be measured is through the *program complexity.* Here the formula τ is fixed and the complexity is measured with respect to the size of Kripke structure only. By Theorem 5 and its proof the program complexity of RegCTL model checking is in NLOGSPACE. As the program complexity of CTL model checking is NLOGSPACE-complete [9], we have that program complexity of RegCTL model checking is NLOGSPACE-complete too. Figure 4 summarises the complexity of both RegCTL and *det*-RegCTL model checking.

	overall complexity	program complexity
det-RegCTL	P-complete	NLOGSPACE-complete
RegCTL	PSPACE-complete	NLOGSPACE-complete

Fig. 4. The complexity of RegCTL and *det*-RegCTL model checking

7 CONCLUSIONS

We studied an extension of the branching time logic CTL with regular expressions. We defined a new branching time logic RegCTL that extends CTL with regular expressions. The model checking problem for RegCTL is PSPACE-complete. However, we identify a large family of RegCTL formulas (including e.g. whole RCTL) that can be checked in P. For constructing the model checking algorithm we adopted the automata-theoretic approach which allows for an effective distribution.

REFERENCES

[1] DiVinE – Distributed Verification Environment. `http://anna.fi.muni.cz/divine`.

[2] Beer, I.—Ben-David, S.—Eisner, C.—Fisman, D.—Gringauze, A.—Rodeh, Y.: The Temporal Logic Sugar. In Proceedings of CAV '01, Vol. 2102 of Lecture Notes in Computer Science, pp. 363–367, Springer, 2001.

[3] Beer, I.—Ben-David, S.—Landver, A.: On-the-Fly Model Checking of RCTL Formulas. In Proceedings of CAV '98, Vol. 1427 of Lecture Notes in Computer Science, pp. 184–194, Springer, 1998.

[4] Ben-Ari, M.—Manna, Z.—Pnueli, A.: The Temporal Logic of Branching Time. Acta Informatica, Vol. 20, 1983, pp. 207–226.

[5] Bollig, B.—Leucker, M.—Weber, M.: Local Parallel Model Checking for the Alternation-Free Mu-Calculus. In Proceedings of the 9th International SPIN Workshop on Model Checking of Software (SPIN '02), Vol. 2318 of Lecture Notes in Computer Science, Springer-Verlag Inc., 2002.

[6] Hamaguchi, K.—Hiraishi, H.—Yajima, S.: Branching Time Regular Temporal Logic for Model Checking with Linear Time Complexity. In Proceedings of CAV '90, Vol. 531 of Lecture Notes in Computer Science, pp. 253–262, Springer, 1991.

[7] Henriksen, J.—Thiagarajan, P.: Dynamic Linear Time Temporal Logic. Annals of Pure and Applied Logic, Vol. 96, 1999, Nos. 1–3, pp. 187–207.

[8] Kupferman, O.—Piterman, N.—Vardi, M.: Extended Temporal Logic Revisited. In Proceedings of CONCUR '01, Vol. 2154 of Lecture Notes in Computer Science, pp. 519–535, Springer, 2001.

[9] Kupferman, O.—Vardi, M.—Wolper, P.: An Automata-Theoretic Approach to Branching Time Model Checking. Journal of the ACM, Vol. 47, 2000, No. 2, pp. 312–360.

[10] Mateescu, R.—Sighireanu, M.: Efficient on-the-Fly Model-Checking for Regular Alternation-Free Mu-Calculus. In S. Gnesi, I. Schieferdecker, and A. Rennoch, editors, Proceedings of FMICS '00, GMD Report 91, pp. 65–86, April 2000.

[11] Muller, D. E.—Schupp, P. E.: Alternating Automata on Infinite Trees. Theoretical Computer Science, Vol. 54, 1987, pp. 267–276.

[12] Stirling, C.: Games for Bisimulation and Model Checking. In Notes for Mathfit Workshop on finite model theory, University of Wales, Swansea, 1996.

[13] Vardi, M.—Wolper, P.: Reasoning about Infinite Computations. Information and Computation, Vol. 115, 1994, No. 1, pp. 1–37.

Tomáš Brázdil is a Ph. D. student at Faculty of Informatics, Masaryk University, Brno, Czech Republic. His research focuses on formal verification of probabilistic systems and temporal logic.

Ivana Černá graduated in computer science at Comenius University, Bratislava, Slovakia. Currently she is an Associate Professor at Faculty of Informatics, Masaryk University, Brno, Czech Republic. Her interests are in formal verification and complexity.