

The Decidability of RPTL

Fahad Khan¹

Abstract. In this paper we look at regular path temporal logic, RPTL, a modal logic which combines the ability to quantify over (finite) paths described by regular expressions (a property which characterises PDL) with the addition of temporal operators. The formulation of RPTL was inspired by agent programming verification considerations. In this paper we prove the decidability of RPTL and establish complexity bounds on the satisfiability problem for RPTL by translating it into the theory of alternating tree automata on infinite trees.

1 RPTL

1.1 Introduction

In this paper we look at regular path temporal logic abbreviated as RPTL, a modal logic which combines the ability to quantify over (finite) paths described by regular expressions (a property which characterises PDL) with the addition of temporal operators. RPTL is based on aspects of Claßen and Lakemeyer’s situation calculus variant ESG, as described in [3], and was initially adapted from ESG by Alechina and Logan². Their motivation was to enable the formulation and verification of properties that arise during the executions of agent programs, in particular those properties which hold prior to the termination of a program.

In order to describe this initial motivation in more detail we need to take a brief detour into agent programming territory. In the 2007 paper *A Logic of Agent Programs* [1], Alechina et al., were able to define a sound and complete logic for reasoning about agent programs written in the agent programming language SimpleAPL – a language which allows for the implementation of software agents with beliefs, goals, and plans. SimpleAPL’s operational semantics are defined in terms of a transition system, with each transition corresponding to a single execution step between agent configurations.

The sound and complete logic which Alechina et al., defined and into which they translated SimpleAPL programs was based on PDL. In [1] they were able to prove correspondence theorems which allowed them to formally specify various properties of SimpleAPL programs under different execution strategies. While this enabled them to express, for example, goal achievement at the end of some/every program execution, they were unable to check for properties which held while the program was executing.

This signalled the usefulness of expressions such as $\langle\pi\rangle Fp$ where $\langle\pi\rangle$ stands for ‘there exists a path described by a regular expression π ’ and Fp stands for ‘somewhere on this path, p holds’. However in PDL we are limited to using existential modalities in expressions of the form $\langle\pi\rangle\phi$ - there is a path described by π and at the

last state of this path ϕ holds, and in CTL we can use them in expressions of the form EFp - there exists some path and somewhere on this path p holds, without being able to say that this path is described by π . And this led to the adaption and simplification of ESG which resulted in RPTL.

RPTL is very similar to both PDL and CTL, the obvious question was how it stood in relation to both of these logics. Initially an attempt was made to translate RPTL into PDL to show that it was a fragment of that logic. However, there seems to be no obvious or straightforward way of doing so. However, PDL can be easily embedded into RPTL (see below). This gives us an EXPTIME lower bound on the complexity of the satisfiability problem for RPTL. RPTL does not contain CTL, unlike RCTL which is an extension of CTL with regular expressions [2], and ECTL which is also an extension of CTL [4].

The central aim of this report is to prove the decidability of RPTL. We achieve this by translating each formula of RPTL into an alternating tree automaton on infinite trees (ATA) and making use of the fact that the non emptiness problem for ATA is in EXPTIME. The translations are based on Thomas Wilke’s translation of the μ -calculus into ATA in [8] (also see [7] for another translation this time of CTL into alternating tree automata). So that RPTL is a fragment of L_μ , though a direct translation from RPTL to L_μ is difficult.

As with PDL we can add tests to our RPTL program expressions. For the sake of simplification we do not add them, but the proof for RPTL with tests is a straightforward extension of the current proof.

RPTL is related to Process Logic. Process Logics also allow us to quantify over paths and to talk about what happens during them – see especially Vardi and Wolper’s YAPL [6] and PL [5] both of which extend PDL by allowing us to talk about what happens within a path. However these logics tend to be slightly too strong for our purposes with higher complexity than is desirable.

As for the rest of this paper. We start in Section 1.2. with a definition of the syntax and semantics of RPTL and in Section 1.3 we give a definition of ATA’s and MAR’s. In Section 2.1 we give a basic outline of the proof of decidability given in the rest of the paper.

1.2 Syntax and Semantics of RPTL

RPTL has four temporal operators, two of them are unary next operators corresponding to the next operators of CTL but labelled with a regular expression, $[\pi]X$, $\langle\pi\rangle X$, the other two are ternary ‘until’ (three place until) operators, box until $[\pi]\mathcal{U}T$ and diamond until $\langle\pi\rangle\mathcal{U}T$. The intuition behind until $p\mathcal{U}qTr$, for some triple of proposition variables, p, q, r , is that given a path $x = (w_0, \dots, w_k)$ we want to determine whether either the proposition p holds in the path x until q for the subpath $x1 = (w_0, \dots, w_{k-1})$, or p holds at each point in $x1$ and r holds at w_k . We introduce the until because it is a natural generalisation of the until temporal operator for finite paths,

¹ University of Nottingham, England, email: afk@cs.nott.ac.uk

² Personal Communication.

where we know that there is an end point, and because it simplifies the proof of decidability.

Definition 1 Given a fixed, infinite, set of proposition symbols, $\Phi = \{p, q, \dots\}$, we can inductively define the set of RPTL formulae as follows:

- $\perp, p \in \Phi$ are both formulae of RPTL,
- if ϕ, ψ are formulae of RPTL then so are $\neg\phi$ and $(\phi \wedge \psi)$,
- if ϕ, ψ and χ are formulae of RPTL and π is a program expression, as defined below, then $[\pi]X\phi, \langle\pi\rangle X\phi, [\pi]\phi\mathcal{U}\psi\mathcal{T}\chi, \langle\pi\rangle\phi\mathcal{U}\psi\mathcal{T}\chi$ are also formulae of RPTL.

Assuming a fixed set of basic programs $\Psi_0 = \{a, b, \dots\}$ we inductively construct the set of program expressions, Ψ , used in the previous definition, in the following manner:

- each basic program $a \in \Psi_0$ is a program,
- if α and β are programs, then so too are $\alpha \cup \beta, \alpha; \beta$, and α^* .

Definition 2 Let $\mathcal{M}$ be a structure, $\mathcal{M} = (W, \tau, \kappa)$, then $\mathcal{M}$ is a model for RPTL if:

- W is a set of states,
- $\tau(a) \subseteq (W \times W)$ gives us the set of state transitions for $a \in \Psi_0$. We can extend this inductively to give us a set of paths $\tau(\pi) \subseteq (W \times W)^*$ corresponding to any program expression π in M :
 - $\tau(\pi_1 \cup \pi_2) = \{z : z \in \tau(\pi_1) \cup \tau(\pi_2)\}$,
 - $\tau(\pi_1; \pi_2) = \{z_1 \circ z_2 : z_1 \in \tau(\pi_1), z_2 \in \tau(\pi_2)\}$, where $\circ$ is a concatenation of paths operator,
 - $\tau(\pi_1^*)$ is the set of all paths consisting of zero or finitely many concatenations of paths in $\tau(\pi_1)$.
- $\kappa(p) \subseteq W$ is a function that for each $p \in \Phi$ gives us the set of states in W at which p holds. We can extend κ to map formulae ϕ to sets of states $\kappa(\phi)$ in W in the usual way.

Assume state $w \in W$ and that ϕ, ψ , and χ are RPTL formulae and π is a program expression. The relation $\models$ of a formula being true in a state of a model is defined inductively as follows:

- $(\mathcal{M}, w) \models p$ iff $w \in \kappa(p)$,
- $(\mathcal{M}, w) \models \neg\phi$ iff $\mathcal{M}, w \not\models \phi$
- $\mathcal{M}, w \models (\phi \wedge \psi)$ iff $\mathcal{M}, w \models \phi$ and $\mathcal{M}, w \models \psi$,
- $(\mathcal{M}, w) \models [\pi]X\phi$ iff for all paths $x = (w_0, \dots, w_l) \in \tau(\pi)$ where $w_0 = w$ we have that $\mathcal{M}, w_l \models \phi$,
- $(\mathcal{M}, w) \models \langle\pi\rangle X\phi$ iff there exists a path $x = (w_0, \dots, w_l) \in \tau(\pi)$ such that $w_0 = w$ and $\mathcal{M}, w_l \models \phi$,
- $(\mathcal{M}, w) \models [\pi]\phi\mathcal{U}\psi\mathcal{T}\chi$ iff for all paths $x = (w_0, \dots, w_l) \in \tau(\pi)$ such that $w_0 = w$ we have that either
 1. for some initial prefix $(w_0, \dots, w_k)$ of x we have that $\mathcal{M}, w_k \models \psi$ and $\mathcal{M}, w_i \models \phi$ for $i < k < l$, or
 2. $\mathcal{M}, w_i \models \phi$ for all $i < l$ and $\mathcal{M}, w_l \models \chi$.
- $(\mathcal{M}, w) \models \langle\pi\rangle\phi\mathcal{U}\psi\mathcal{T}\chi$ iff there exists a path $x = (w_0, \dots, w_l) \in \tau(\pi)$ where $w_0 = w$ such that either
 1. for some initial prefix $(w_0, \dots, w_k)$ of x we have that and $\mathcal{M}, w_k \models \psi$ and $\mathcal{M}, w_i \models \phi$ for $i < k < l$, or
 2. $\mathcal{M}, w_i \models \phi$ for all $i < l$ and $\mathcal{M}, w_l \models \chi$.

It is clear from the preceding definitions that we have the following equivalences:

- $[\pi]Xp \equiv \neg\langle\pi\rangle X\neg p$,
- $\langle\pi\rangle Xp \equiv \neg[\pi]X\neg p$,
- $[\pi]p\mathcal{U}q\mathcal{T}r \equiv \neg\langle\pi\rangle\neg q\mathcal{U}(\neg p \wedge \neg q)\mathcal{T}\neg r$,
- $\langle\pi\rangle p\mathcal{U}q\mathcal{T}r \equiv \neg[\pi]\neg q\mathcal{U}(\neg p \wedge \neg q)\mathcal{T}\neg r$,

We can easily define a binary until operator $\mathcal{U}$ corresponding to the temporal until operator, using the untiltil operator, so that for example $[\pi]p\mathcal{U}q$ is shorthand for $[\pi]p\mathcal{U}q\mathcal{T}q$. We can then define the F operator by translating formulae such as $\langle\pi\rangle Fp$ as $\langle\pi\rangle\mathcal{T}\mathcal{U}p$.

The embedding of PDL into RPTL is also straightforward, so for example formulae of the form $\langle\pi\rangle p$ can be written as $\langle\pi\rangle\mathcal{T}\mathcal{U}\perp p$.

1.3 Alternating Tree Automata

Definition 3 An alternating tree automaton (ATA) is a tuple $\mathcal{A} = (S, s_I, \delta, \Psi, \Omega)$ such that S is a finite set of states, $s_I \in S$ is an initial state, δ is a transition function defined below, Π is a finite set of path labels, and $\Omega : S \rightarrow \omega$ is a function that assigns a whole number priority to each state

We define the transition function as follows:

Definition 4 The transition function δ maps each state $s \in S$ to a transition condition over S , where the set of all transition conditions over S is defined by:

- 0 and 1 are transition conditions over S ,
- p and $\neg p$ are transition conditions over S for every $p \in \Phi$,
- $s, [a]s$, and $\langle a \rangle s$ are transition conditions over S , for every $s \in S$ and $a \in \Psi$,
- $s \wedge s'$ and $s \vee s'$ are transition conditions over S , for $s, s' \in S$.

Note that we are dealing exclusively with ATA which accept infinite trees (although they will also accept finite trees too). We have a notion of run specific to ATA's:

Definition 5 A run of $\mathcal{A}$ on a pointed Kripke structure³ $(\mathcal{M}, w)$ where $\mathcal{M} = (W, \tau, \kappa)$, is a $(W \times S)$ -vertex-labeled tree $r = (V^r, E^r, \lambda^r)$ such that the root ρ^r is labeled (w_0, s_I) by the function λ^r and for every vertex v with label (w, s) assigned by λ^r the following conditions are satisfied, we write this as λ when no confusion is possible. In the following definition we will use the notation $\Delta_r(v)$ to refer to the E -successors of the vertex v in W , and $\Delta_a(v)$ to refer to the a -successors of v in $\mathcal{M}$:

- $\delta(s) \neq 0$,
- if $\delta(s) = q$, then $w \in \kappa(q)$, and if $\delta(s) = \neg q$, then $w \notin \kappa(q)$,
- if $\delta(s) = s'$, then there exists $v' \in \Delta_r(v)$ such that $\lambda(v') = (w, s')$,
- if $\delta(s) = \langle a \rangle s'$, then there exists $v' \in \Delta_r(v)$ such that $s^r(v') = s'$ and $w^r(v') \in \Delta_a(w)$,
- if $\delta(s) = [a]s'$, then for every $w' \in \Delta_a(w)$ there exists $v' \in \Delta_r(v)$ such that $\lambda(v') = (w', s')$,
- if $\delta(s) = s' \vee s''$, then there exist $v', v'' \in \Delta_r(v)$ such that $\lambda(v') = (w, s')$ or $\lambda(v'') = (w, s'')$,
- if $\delta(s) = s' \wedge s''$, then there exist $v', v'' \in \Delta_r(v)$ such that $\lambda(v') = (w, s')$ and $\lambda(v'') = (w, s'')$.

We will only be interested in minimal run trees in the paper, in which case the conditions for any vertex v with label (w, s) are altered as follows:

³ A pointed Kripke structure is a pair $(\mathcal{M}, w)$, where $\mathcal{M} = (W, \tau, \kappa)$, and $w \in W$.

- $\delta(s) \neq 0$,
- if $\delta(s) = q$, then $w \in \kappa(q)$, and if $\delta(s) = \neg q$, then $w \notin \kappa(q)$,
- if $\delta(s) = s'$, then there exists $v' \in \Delta_r(v)$ such that $\lambda(v') = (w, s')$, and there is only one such successor,
- if $\delta(s) = \langle a \rangle s'$, then there exists $v' \in \Delta_r(v)$ such that $s^r(v') = s'$ and $w^r(v') \in \Delta_a(w)$, and there is only one such successor,
- if $\delta(s) = [a]s'$, then for every $w' \in \Delta_a(w)$ there exists $v' \in \Delta_r(v)$ such that $\lambda(v') = (w', s')$, and for each $w' \in \Delta_a(w)$ there is exactly one successor labelled with (w', s') .
- if $\delta(s) = s' \vee s''$, then there exist $v', v'' \in \Delta_r(v)$ such that $\lambda(v') = (w, s')$ or $\lambda(v'') = (w, s'')$, and there is only one such successor,
- if $\delta(s) = s' \wedge s''$, then there exist $v', v'' \in \Delta_r(v)$ such that $\lambda(v') = (w, s')$ and $\lambda(v'') = (w, s'')$, and we assume that there are exactly two such successors.

For each vertex $v = (w, s)$ of r , we use v^W to denote the first component of v and v^S to refer to the second component of v , so that $v^W = w$ and $v^S = s$.

Also with minimal run trees r we impose the condition that for any vertex u with state component s if u has an edge leading from it, then the transition condition $\delta(s)$ contains one or more states.

Note that in the rest of this paper we will allow as transition conditions basic multi-modal formulae over Ψ_0, Φ , and S . This does not lead to an increase in expressive power, see [8].

We call an infinite branch b of r accepting if the largest value in X , the set of whole numbers n such that n is allocated to the state component of a vertex $v \in W$ by Ω for an infinite number of vertices, is even. A run r is accepting if all infinite branches are accepting. (If there are no infinite branches then the run is trivially accepting).

We use the acronym MAR to refer to a minimal accepting run.

We denote the set of pointed Kripke structures accepted by an ATA $\mathcal{A}$ by $\|\mathcal{A}\|$.

And finally we have:

Definition 6 Given an ATA $\mathcal{A}$ we can derive the transition graph $\mathcal{G}(\mathcal{A})$ of $\mathcal{A}$ namely the directed graph whose vertex set is the set of states of $\mathcal{A}$ where we add an edge between states s and s' iff s' appears in $\delta(s)$.

2 The Decidability of RPTL

We can ‘translate’ any formula of RPTL into an ATA enabling us to prove the decidability of RPTL. In other words for any formula ϕ of RPTL we can find an ATA $\mathcal{A}(\phi)$ such that for any pointed Kripke structure $(\mathcal{M}, w)$, $(\mathcal{M}, w) \models \phi$ iff $(\mathcal{M}, w) \in \|\mathcal{A}(\phi)\|$.

2.1 Outline of Proof

We start off by translating limited fragments of the language into ATA’s. So that for formulae of the form $\llbracket \pi \rrbracket Xp$, $\langle \pi \rangle Xp$, $\llbracket \pi \rrbracket p\mathcal{U}qTr$, and $\langle \pi \rangle p\mathcal{U}qTr$, where π is a regular expression and p, q and r are propositional variables, we define corresponding ATA’s of the form $\mathcal{D}[\llbracket \pi \rrbracket Xp]$, $\mathcal{D}[\langle \pi \rangle Xp]$, $\mathcal{D}[\llbracket \pi \rrbracket p\mathcal{U}qTr]$, and $\mathcal{D}[\langle \pi \rangle p\mathcal{U}qTr]$. So that for all pointed Kripke structures $(\mathcal{M}, w)$, a RPTL formula satisfies $(\mathcal{M}, w)$ iff $(\mathcal{M}, w)$ belongs to its corresponding ATA, e.g., for each pointed Kripke structure $(\mathcal{M}, w) \in \mathcal{D}[\llbracket \pi \rrbracket Xp]$ iff $(\mathcal{M}, w) \in \|\mathcal{D}[\llbracket \pi \rrbracket Xp]\|$.

In Section 2.2 we define ATA’s $\mathcal{D}[\llbracket \pi \rrbracket Xp]$ and $\mathcal{D}[\langle \pi \rangle Xp]$ such that the translation is clearly correct.

In Section 2.3 we define a new kind of ATA, the Double Alternating Tree Automaton (DATA). DATA’s keep track of a so called final state, and also have two states corresponding to a pair of propositional variables p, q .

We then construct for any regular expression π and pair of propositional variables p, q , a DATA $\mathcal{D}[\llbracket \pi \rrbracket^- p\mathcal{U}q]$ such that for any pointed Kripke structure $(\mathcal{M}, w)$ where $\mathcal{M} = (W, \tau, \kappa)$, the DATA $\mathcal{D}[\llbracket \pi \rrbracket^- p\mathcal{U}q]$ enables us to keep a track of all paths in $x = (w_0, \dots, w_k) \in \tau(\pi)$ – or at least their first and last points – such that either $w_i \models p \wedge \neg q$ for all $i < k$ or $w_j \models \neg p \wedge \neg q$ and $w_i \models p \wedge \neg q$.

This will enable us to define, given one extra propositional variable r , an ATA $\mathcal{D}[\llbracket \pi \rrbracket p\mathcal{U}qTr]$ for which given any pointed Kripke structure $(\mathcal{M}, w)$ satisfies the RPTL expression $\llbracket \pi \rrbracket p\mathcal{U}qTr$ if and only if $(\mathcal{M}, w) \in \|\mathcal{D}[\llbracket \pi \rrbracket p\mathcal{U}qTr]\|$, thereby relating an untiltil formula of the form $\llbracket \pi \rrbracket p\mathcal{U}qTr$ to a corresponding ATA $\mathcal{D}[\llbracket \pi \rrbracket p\mathcal{U}qTr]$.

In Section 2.4 we state a corresponding theorem for untiltil formulae of the form $\langle \pi \rangle p\mathcal{U}qTr$ and their corresponding ATA’s $\mathcal{D}[\langle \pi \rangle p\mathcal{U}qTr]$.

Finally in Section 2.5 we put the results from the previous few sections together. For any formula ϕ of RPTL we can find an equivalent formula ϕ' of $RCTL$. We then define an ATA $\mathcal{A}(\phi)$ which is the translation of ϕ' . Since negation of ATA’s can lead to an exponential blowup, we are careful to define a special ATA $\mathcal{N}(\phi)$ which we use in the definition of $\mathcal{A}(\phi)$.

2.2 $\mathcal{D}[\llbracket \pi \rrbracket Xp]$, $\mathcal{D}[\langle \pi \rangle Xp]$

We first define a function ζ that takes a formula of the form $\llbracket \pi \rrbracket Xs$, or $\langle \pi \rangle Xs$, where $s \in S \cup \Phi$ and returns a basic multi-modal formula over S and Ψ_0 and Φ .

Definition 7 For any regular expression π we define $\zeta(\llbracket \pi \rrbracket Xs)$ as follows (by induction on π):

- $\zeta(\llbracket a \rrbracket Xs) = [a]s$,
- $\zeta(\llbracket \pi_1; \pi_2 \rrbracket Xs) = \zeta(\llbracket \pi_1 \rrbracket Xs)$ where $(\pi_1 \neq \varepsilon$ and ε is the empty string),
- $\zeta(\llbracket \pi_1 \cup \pi_2 \rrbracket Xs) = \zeta(\llbracket \pi_1 \rrbracket Xs) \wedge \zeta(\llbracket \pi_2 \rrbracket Xs)$,
- $\zeta(\llbracket \pi_1^* \rrbracket Xs) = s \wedge \zeta(\llbracket \pi_1 \rrbracket Xs)$.

Similarly for any regular expression π we define $\zeta(\langle \pi \rangle Xs)$ as follows (by induction on π):

- $\zeta(\langle a \rangle Xs) = \langle a \rangle s$,
- $\zeta(\langle \pi_1; \pi_2 \rangle Xs) = \zeta(\langle \pi_1 \rangle Xs)$ where $(\pi_1 \neq \varepsilon)$,
- $\zeta(\langle \pi_1 \cup \pi_2 \rangle Xs) = \zeta(\langle \pi_1 \rangle Xs) \vee \zeta(\langle \pi_2 \rangle Xs)$,
- $\zeta(\langle \pi_1^* \rangle Xs) = s \vee \zeta(\langle \pi_1 \rangle Xs)$.

It’s clear that for any pointed Kripke structure $(\mathcal{M}, w_0)$ we have

$$(\mathcal{M}, w_0) \models \llbracket \pi \rrbracket Xp \text{ iff } (\mathcal{M}, w_0) \models \zeta(\llbracket \pi \rrbracket Xp),$$

and

$$(\mathcal{M}, w_0) \models \langle \pi \rangle Xp \text{ iff } (\mathcal{M}, w_0) \models \zeta(\langle \pi \rangle Xp).$$

Then it is easy to construct the ATA $\mathcal{D}[\llbracket \pi \rrbracket Xp] = (S, s_I, \delta, \Omega)$. Let $S = \{s, s_p\}$, $s_I = s$, and $\Omega(s) = \Omega(s_p) = 1$. We define $\delta(s) = \zeta(\llbracket \pi \rrbracket Xs_p)$ and $\delta(s_p) = p$. The only difference in the definition of $\mathcal{D}[\langle \pi \rangle Xp] = (S, s_I, \delta, \Omega)$ is that $\delta(s) = \zeta(\langle \pi \rangle Xs_p)$.

Lemma 1 Let $\mathcal{D}[\llbracket \pi \rrbracket Xp] = (S, s_I, \delta, \Omega)$ be an ATA, where $p \in \Phi$, and $(\mathcal{M}, w)$ be a pointed Kripke structure with $w_0 = w$ then:

$$(\mathcal{M}, w_0) \models \llbracket \pi \rrbracket Xp \text{ if and only if } (\mathcal{M}, w_0) \in \|\mathcal{D}[\llbracket \pi \rrbracket Xp]\|.$$

Proof. omitted due to limited space $\square$

Similarly the following lemma also holds:

Lemma 2 Let $\mathcal{D}(\|\pi\|)Xp = (S, s_I, \delta, \Omega)$ be an ATA, where $p \in \Phi$, and $(\mathcal{M}, w)$ be a pointed Kripke structure with $w_0 = w$ then:

$$(\mathcal{M}, w_0) \models \langle \|\pi\| \rangle Xp \text{ if and only if } (\mathcal{M}, w_0) \in \|\mathcal{D}(\|\pi\|)Xp\|.$$

2.3 $\mathcal{D}[\|\pi\|]p\mathcal{M}q\mathcal{T}r$

We will define a new kind of alternating tree automaton, the double-alternating tree automaton.

Definition 8 A Double-alternating tree automaton (DATA) for any pair of proposition symbols p, q is a tuple $\mathcal{D}(p, q) = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ where $\mathbf{R}$ is a disjoint pair of finite sets of states $\mathbf{R} = (R^1, R^2)$; $\mathbf{r}_I = (r_I^1, r_I^2)$ is a pair of states, where $r_I^1 \in R^1$ is the initial state of the automaton, and $r_I^2 \in R^2$ is a special marked state; $\mathbf{r}_f = (r_f^1, r_f^2)$ is a pair of states, where $r_f^1 \in R^1$ and $r_f^2 \in R^2$ are marked states.

We also single out four more states $r_p, r_{\neg p}, r_q, r_{\neg q} \in R^1$. Additionally δ is a transition function that maps each state $r \in R^1 \cup R^2$ to a complex transition condition over $R^1 \cup R^2$. Finally $\Omega : (R^1 \cup R^2) \rightarrow \omega$ is a function that assigns a whole number priority to each state in $R^1 \cup R^2$.

A run of the DATA $\mathcal{D}(p, q)$ on a pointed Kripke Structure $(\mathcal{M}, w)$ is simply a run of the ATA $\mathcal{A}(p, q) = (R^1 \cup R^2, r_I^1, \delta, \Omega)$ on $(\mathcal{M}, w)$.

DATA's are mere notational conveniences and we can view a DATA $\mathcal{D} = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as essentially being a ATA, $\mathcal{F}(\mathcal{D}) = (S, s_I, \delta', \Omega')$ where each $s \in S$ is a tuple $(s', s_p, s_{\neg p}, s_q, s_{\neg q}, s_f)$, where $s' \in R^1 \cup R^2$, and the other components are used to determine whether, for example $s' = s_f$. We alter the transition function δ accordingly

We can also easily alter the runs of the DATA's to get the runs of their equivalent ATAs.

We construct the DATA $\mathcal{D}[\|\pi\|]^{-}p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$, for a program expression π , and a fixed pair of propositional variables p, q , by induction:

- (Base case: $\pi = a$) For any $a \in \Psi_0$ we define the DATA $\mathcal{D}[\|a\|]^{-}p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ where $\mathbf{R} = (\{s_0, s_1, s_2, s_3, s_4, s_5\}, \{t_0, t_1\})$, Ω assigns 1 to all states, and the assignments are: $\mathbf{r}_I = (s_0, t_0)$, $\mathbf{r}_f = (s_1, t_1)$, $r_p = s_2$, $r_{\neg p} = s_3$, $r_q = s_4$ and $r_{\neg q} = s_5$.

The transition function δ is defined as follows:

- $\delta(r_I^1) = r_q \vee (r_p \wedge r_{\neg q} \wedge [a]r_f^1) \vee (r_{\neg p} \wedge r_{\neg q} \wedge r_I^2)$,
- $\delta(r_f^1) = 1$,
- $\delta(r_q) = q$,
- $\delta(r_{\neg q}) = \neg q$
- $\delta(r_{\neg p}) = \neg p$
- $\delta(r_p) = p$,
- $\delta(r_I^2) = [a]r_f^2$,
- $\delta(r_f^2) = 1$.

- (Inductive case: $\pi = \pi_1; \pi_2$) Here we assume we have already defined the DATA's $\mathcal{D}[\|\pi_1\|]^{-}p\mathcal{M}q = (\mathbf{S}, \mathbf{s}_I, \mathbf{s}_f, s_p, s_{\neg p}, s_q, s_{\neg q}, \delta_{\mathbf{S}}, \Omega_{\mathbf{S}})$, $\mathcal{D}[\|\pi_2\|]^{-}p\mathcal{M}q =$

$(\mathbf{T}, \mathbf{t}_I, \mathbf{t}_f, t_p, t_{\neg p}, t_q, t_{\neg q}, \delta_{\mathbf{T}}, \Omega_{\mathbf{T}})$ for the program expressions π_1 and π_2 .

Then we can construct $\mathcal{D}[\|\pi_1; \pi_2\|]^{-}p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as follows. We assume S^1 and T^1 are disjoint and set R^1 to be their union; similarly we assume that S^2 and T^2 are disjoint and let R^2 be their union. Set $\mathbf{r}_I = \mathbf{s}_I, \mathbf{r}_f = \mathbf{t}_f$, and $r_p = s_p, r_{\neg p} = s_{\neg p}, r_q = s_q, r_{\neg q} = s_{\neg q}$. Let $\Omega(r) = \Omega_{\mathbf{S}}(r)$ for $r \in S^1 \cup S^2$, $\Omega(r) = \Omega_{\mathbf{T}}(r)$ for $r \in T^1 \cup T^2$. Finally the transition function $\delta(r)$ for $r \in R^1 \cup R^2$, is defined as follows:

$$\delta(r) = \begin{cases} \delta_{\mathbf{S}}(r) & \text{if } r \in S^1 \cup S^2 - \{s_f^1, s_f^2\}, \\ \delta_{\mathbf{T}}(r) & \text{if } r \in T^1 \cup T^2, \\ t_f^1 & \text{if } r = s_f^1, \\ t_f^2 & \text{if } r = s_f^2. \end{cases}$$

and for each formula $\delta(r)$, we replace each instance of s_p or t_p by r_p , similarly s_q or t_q are replaced by r_q , $s_{\neg p}$ or $t_{\neg p}$ are replaced by $r_{\neg p}$ and $s_{\neg q}$ or $t_{\neg q}$ are replaced by $r_{\neg q}$.

- (Inductive case: $\pi = \pi_1 \cup \pi_2$) Again we assume we have already defined the DATA's $\mathcal{D}[\|\pi_1\|]^{-}p\mathcal{M}q = (\mathbf{S}, \mathbf{s}_I, \mathbf{s}_f, s_p, s_{\neg p}, s_q, s_{\neg q}, \delta_{\mathbf{S}}, \Omega_{\mathbf{S}})$, $\mathcal{D}[\|\pi_2\|]^{-}p\mathcal{M}q = (\mathbf{T}, \mathbf{t}_I, \mathbf{t}_f, t_p, t_{\neg p}, t_q, t_{\neg q}, \delta_{\mathbf{T}}, \Omega_{\mathbf{T}})$ for the program expressions π_1 and π_2 .

Then we construct $\mathcal{D}[\|\pi_1 \cup \pi_2\|]^{-}p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as follows. We assume S^1 and T^1 are disjoint and set R^1 to be their union with two new states r_1 and r_2 ; and let R^2 be the disjoint union of S^2 and T^2 with two new states r_3 and r_4 . We set $\mathbf{r}_I = (r_1, r_3)$, and $\mathbf{r}_f = (r_2, r_4)$. We set $\Omega(r_1), \Omega(r_2), \Omega(r_3), \Omega(r_4)$ to be 1, the rest of the states as above. The transition function $\delta(r)$ for $r \in R^1 \cup R^2$, is defined as follows:

$$\delta(r) = \begin{cases} \delta_{\mathbf{S}}(r) & \text{if } r \in S^1 \cup S^2 - \{s_f^1, s_f^2\}, \\ \delta_{\mathbf{T}}(r) & \text{if } r \in T^1 \cup T^2 - \{t_f^1, t_f^2\}, \\ r_f^1 & \text{if } r = s_f^1 \text{ or } r = t_f^1, \\ ((r_{\neg p} \wedge r_{\neg q}) \wedge r_I^2) \vee (r_p \wedge s_I^1 \wedge t_I^1) & \text{if } r = r_I^1, \\ r_f^2 & \text{if } r = s_f^2 \text{ or } r = t_f^2, \\ s_I^2 \wedge t_I^2 & \text{if } r = r_I^2, \\ 1 & \text{if } r = r_f^1 \text{ or } r = r_f^2. \end{cases}$$

and for each formula $\delta(r)$, we replace each instance of s_p or t_p by r_p , similarly s_q or t_q are replaced by r_q , $s_{\neg p}$ or $t_{\neg p}$ are replaced by $r_{\neg p}$ and $s_{\neg q}$ or $t_{\neg q}$ are replaced by $r_{\neg q}$.

- (Inductive case: $\pi = \pi_1^*$) We assume that we have already defined the DATA $\mathcal{D}[\|\pi_1\|]^{-}p\mathcal{M}q = (\mathbf{S}, \mathbf{s}_I, \mathbf{s}_f, s_p, s_{\neg p}, s_q, s_{\neg q}, \delta_{\mathbf{S}}, \Omega_{\mathbf{S}})$ for the regular expression π_1 .

Then we can construct $\mathcal{D}[\|\pi_1^*\|]^{-}p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as follows. We let R^1 be the disjoint union of S^1 with two new states r_1 and r_2 , and R^2 be the disjoint union of S^2 with two new states r_3 and r_4 . We set $\mathbf{r}_I = (r_1, r_2)$ and $\mathbf{r}_f = (r_3, r_4)$. We set $\Omega(r_1), \Omega(r_2), \Omega(r_3), \Omega(r_4)$ to be 1 and $\Omega(s_f^1) = \Omega(s_f^2) = 2$ and the rest as above. The transition function $\delta(r)$ for $r \in R^1 \cup R^2$, is defined as follows:

$$\delta(r) = \begin{cases} \delta_{\mathbf{S}}(r) & \text{if } r \in S^1 \cup S^2 - \{s_f^1, s_f^2\} \\ ((r_{\neg p} \wedge r_{\neg q}) \wedge r_I^2) \vee (r_p \wedge s_I^1 \wedge r_f^1) & \text{if } r = r_I^1, \\ s_I^1 \wedge r_f^1 & \text{if } r = s_f^1, \\ s_I^2 \wedge r_f^2 & \text{if } r = r_I^2 \text{ or } r = s_f^2, \\ 1 & \text{if } r = r_f^1 \text{ or } r = r_f^2. \end{cases}$$

and for each formula $\delta(r)$, we replace each instance of s_p or t_p by r_p , similarly s_q or t_q are replaced by r_q , $s_{\neg p}$ or $t_{\neg p}$ are replaced by $r_{\neg p}$ and $s_{\neg q}$ or $t_{\neg q}$ are replaced by $r_{\neg q}$.

Proposition 1 For any pointed Kripke structure $(\mathcal{M}, w)$, there ex-

ists exactly one MAR of $\mathcal{D}[\llbracket \pi \rrbracket]^-p\mathcal{M}q$ on $(\mathcal{M}, w)$ (up to isomorphism).

Lemma 3 Let $\mathcal{D}[\llbracket \pi \rrbracket]^-p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta_{\mathbf{R}}, \Omega_{\mathbf{R}})$ be a DATA, and $\mathcal{D}[\llbracket \pi \rrbracket]p\mathcal{M}qTr$ be the corresponding ATA, where $p, q, r \in \Psi_0$, and $(\mathcal{M}, w)$ be a pointed Kripke structure with $w_0 = w$ then:

$(\mathcal{M}, w_0) \models \llbracket \pi \rrbracket p\mathcal{M}qTr$ if and only if $(\mathcal{M}, w_0) \in \|\mathcal{D}[\llbracket \pi \rrbracket]p\mathcal{M}qTr\|$.

2.4 $\mathcal{D}[\llbracket \pi \rrbracket]p\mathcal{M}qTr$

- (Base case: $\pi = a$) For any $a \in \Psi_0$ we define the DATA $\mathcal{D}[\llbracket a \rrbracket]^-p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ where $\mathbf{R} = (\{s_0, s_1, s_2, s_3, s_4, s_5\}, \{t_0, t_1\})$, Ω assigns 1 to all states, and the assignments are: $\mathbf{r}_I = (s_0, t_0)$, $\mathbf{r}_f = (s_1, t_1)$, $r_p = s_2$, $r_{\neg p} = s_3$, $r_q = s_4$ and $r_{\neg q} = s_5$.

The transition function δ is defined as follows:

- $\delta(r_I^1) = (r_q \wedge r_I^2) \vee (r_p \wedge r_{\neg q} \wedge \langle a \rangle r_f^1)$,
- $\delta(r_f^1) = 1$,
- $\delta(r_q) = q$,
- $\delta(r_p) = p$,
- $\delta(r_{\neg q}) = \neg q$
- $\delta(r_I^2) = \langle a \rangle r_f^2$,
- $\delta(r_f^2) = 1$.

- (Inductive case: $\pi = \pi_1; \pi_2$)

Here we assume we have already defined the DATA's $\mathcal{D}[\llbracket \pi_1 \rrbracket]^-p\mathcal{M}q = (\mathbf{S}, \mathbf{s}_I, \mathbf{s}_f, s_p, s_{\neg p}, s_q, s_{\neg q}, \delta_{\mathbf{S}}, \Omega_{\mathbf{S}})$, $\mathcal{D}[\llbracket \pi_2 \rrbracket]^-p\mathcal{M}q = (\mathbf{T}, \mathbf{t}_I, \mathbf{t}_f, t_p, t_{\neg p}, t_q, t_{\neg q}, \delta_{\mathbf{T}}, \Omega_{\mathbf{T}})$ for the regular expressions π_1 and π_2 .

Then we can construct $\mathcal{D}[\llbracket \pi \rrbracket]^-p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as follows. We assume S^1 and T^1 are disjoint and let R^1 be their union; similarly we assume that S^1 and T^2 are disjoint and let R^2 be their union. Set $\mathbf{r}_I = \mathbf{s}_I$, $\mathbf{r}_f = \mathbf{t}_f$, and $r_p = s_p$, $r_{\neg p} = s_{\neg p}$, $r_q = s_q$, $r_{\neg q} = s_{\neg q}$.

Let $\Omega(r) = \Omega_{\mathbf{S}}(r)$ for $r \in S^1 \cup S^2$, $\Omega(r) = \Omega_{\mathbf{T}}(r)$ for $r \in T^1 \cup T^2$. Finally the transition function $\delta(r)$ for $r \in R^1 \cup R^2$, is defined as follows:

$$\delta(r) = \begin{cases} \delta_1(r) & \text{if } r \in S^1 \cup S^2 - \{s_f^1, s_f^2\}, \\ \delta_2(r) & \text{if } r \in T^1 \cup T^2, \\ t_I^1 & \text{if } r = s_f^1, \\ t_I^2 & \text{if } r = s_f^2. \end{cases}$$

and for each formula $\delta(r)$, we replace each instance of s_p or t_p by r_p , similarly s_q or t_q are replaced by r_q , $s_{\neg p}$ or $t_{\neg p}$ are replaced by $r_{\neg p}$ and $s_{\neg q}$ or $t_{\neg q}$ are replaced by $r_{\neg q}$.

- (Inductive case: $\pi = \pi_1 \cup \pi_2$) Again we assume that we have already defined the DATA's

$$\begin{aligned} \mathcal{D}[\llbracket \pi_1 \rrbracket]^-p\mathcal{M}q &= (\mathbf{S}, \mathbf{s}_I, \mathbf{s}_f, s_p, s_{\neg p}, s_q, s_{\neg q}, \delta_{\mathbf{S}}, \Omega_{\mathbf{S}}), \\ \mathcal{D}[\llbracket \pi_2 \rrbracket]^-p\mathcal{M}q &= (\mathbf{T}, \mathbf{t}_I, \mathbf{t}_f, t_p, t_{\neg p}, t_q, t_{\neg q}, \delta_{\mathbf{T}}, \Omega_{\mathbf{T}}). \end{aligned}$$

Then we can construct $\mathcal{D}[\llbracket \pi_1 \cup \pi_2 \rrbracket]^-p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as follows.

We assume S^1 and T^1 are disjoint and set R^1 to be their union with two new states r_1 and r_2 ; and let R^2 be the disjoint union of S^2 and T^2 with two new states r_3 and r_4 . We set $\mathbf{r}_I = (r_1, r_3)$, and $\mathbf{r}_f = (r_2, r_4)$. We set $\Omega(r_1), \Omega(r_2), \Omega(r_3), \Omega(r_4)$ to be 1, the rest of the states as above. The transition function $\delta(r)$ for $r \in R^1 \cup R^2$, is defined as follows:

$$\delta(r) = \begin{cases} \delta_1(r) & \text{if } r \in S^1 \cup S^2 - \{s_f^1, s_f^2\}, \\ \delta_2(r) & \text{if } r \in T^1 \cup T^2 - \{t_f^1, t_f^2\}, \\ r_f^1 & \text{if } r = s_f^1 \text{ or } r = s_f^2, \\ (r_q \wedge r_I^2) \vee (r_p \wedge r_{\neg q} \wedge (s_I^1 \vee t_I^1)) & \text{if } r = r_I^1, \\ 1 & \text{if } r = r_f^1, \\ r_f^2 & \text{if } r = s_f^2 \text{ or } r = t_f^2, \\ s_I^2 \vee t_I^2 & \text{if } r = r_I^2, \\ 1 & \text{if } r = r_f^2, \end{cases}$$

and for each formula $\delta(r)$, we replace each instance of s_p or t_p by r_p , similarly s_q or t_q are replaced by r_q , $s_{\neg p}$ or $t_{\neg p}$ are replaced by $r_{\neg p}$ and $s_{\neg q}$ or $t_{\neg q}$ are replaced by $r_{\neg q}$.

- (Inductive case: $\pi = \pi_1^*$) We assume that we have already defined the DATA $\mathcal{D}[\llbracket \pi_1 \rrbracket]^-p\mathcal{M}q = (\mathbf{S}, \mathbf{s}_I, \mathbf{s}_f, s_p, s_{\neg p}, s_q, s_{\neg q}, \delta_{\mathbf{S}}, \Omega_{\mathbf{S}})$ for the regular expression π_1 .

Then we can construct $\mathcal{D}[\llbracket \pi_1^* \rrbracket]^-p\mathcal{M}q = (\mathbf{R}, \mathbf{r}_I, \mathbf{r}_f, r_p, r_{\neg p}, r_q, r_{\neg q}, \delta, \Omega)$ as follows. We let R^1 be the disjoint union of S^1 with two new states r_1 and r_2 , and R^2 be the disjoint union of S^2 with two new states r_3 and r_4 . We set $\mathbf{r}_I = (r_1, r_3)$ and $\mathbf{r}_f = (r_2, r_4)$. We set $\Omega(r_1), \Omega(r_2), \Omega(r_3), \Omega(r_4)$ to be 1 and $\Omega(s_f^1) = \Omega(s_f^2) = 2$ and the rest as above. The transition function $\delta(r)$ for $r \in R^1 \cup R^2$, is defined as follows:

$$\delta(r) = \begin{cases} \delta_1(r) & \text{if } r \in S^1 \cup S^2 - \{r_I^1, r_I^2\} \\ (r_q \wedge r_I^2) \vee (r_p \wedge r_{\neg q} \wedge (s_I^1 \vee r_f^1)) & \text{if } r = r_I^1, \\ s_I^1 \vee r_f^1 & \text{if } r = s_f^1, \\ 1 & \text{if } r = r_f^1, \\ s_I^2 \vee r_f^2 & \text{if } r = r_I^2 \text{ or } r = s_f^2, \\ 1 & \text{if } r = r_f^2. \end{cases}$$

and for each formula $\delta(r)$, we replace each instance of s_p or t_p by r_p , similarly s_q or t_q are replaced by r_q , $s_{\neg p}$ or $t_{\neg p}$ are replaced by $r_{\neg p}$ and $s_{\neg q}$ or $t_{\neg q}$ are replaced by $r_{\neg q}$.

Lemma 4 Let $\mathcal{D}[\llbracket \pi \rrbracket]p\mathcal{M}qTr = (S, s_I, s_p, s_{\neg p}, s_q, s_{\neg q}, s_r, \delta, \Omega)$ be a DATA, where $p, q, r \in \Phi$, and $(\mathcal{M}, w)$ be a pointed Kripke structure with $w_0 = w$ then:

$(\mathcal{M}, w_0) \models \langle \pi \rangle p\mathcal{M}qTr$ if and only if $(\mathcal{M}, w_0) \in \|\mathcal{D}[\llbracket \pi \rrbracket]p\mathcal{M}qTr\|$.

2.5 The Translation of RPTL into Alternating Tree Automata

In the following definition and two propositions we work with the ATA's $\mathcal{A}^1 = (S^1, s_I^1, \delta^1, \Omega^1)$ and $\mathcal{A}^2 = (S^2, s_I^2, \delta^2, \Omega^2)$.

Definition 9 We define the ATA $\mathcal{A}^1 \cdot \mathcal{A}^2 = (S^1 \cup S^2 \cup \{s_I\}, s_I, \delta, \Omega)$, where s_I is a new state, $\delta(s_I) = s_I^1 \wedge s_I^2$, and $\Omega(s_I) = 1$, and $\Omega(s_1) = \Omega^1(s)$ for $s \in S^1$, $\Omega(s) = \Omega^2(s)$ for $s \in S^2$.

We can similarly define $\mathcal{A}^1 + \mathcal{A}^2 = (S^1 \cup S^2 \cup \{s_I\}, s_I, \delta, \Omega)$, where s_I is a new state, $\delta(s_I) = s_I^1 \vee s_I^2$, and $\Omega(s_I) = 1$, and $\Omega(s_1) = \Omega^1(s)$ for $s \in S^1$, $\Omega(s) = \Omega^2(s)$ for $s \in S^2$.

Proposition 2 For any pointed Kripke Structure $(\mathcal{M}, w)$ where $\mathcal{M} = (W, \tau, \kappa)$ $(\mathcal{M}, w) \in \|\mathcal{A}^1 \cdot \mathcal{A}^2\|$ if and only if $(\mathcal{M}, w) \in \|\mathcal{A}^1\|$ and $(\mathcal{M}, w) \in \|\mathcal{A}^2\|$.

Proposition 3 For any pointed Kripke Structure $(\mathcal{M}, w)$ where $\mathcal{M} = (W, \tau, \kappa)$ $(\mathcal{M}, w) \in \|\mathcal{A}^1 + \mathcal{A}^2\|$ if and only if $(\mathcal{M}, w) \in \|\mathcal{A}^1\|$ or $(\mathcal{M}, w) \in \|\mathcal{A}^2\|$.

Definition 10 We define $\mathcal{A}(\phi)$ inductively for any RPTL formula ϕ as follows (for this definition we assume that we can define $\mathcal{N}(\chi)$ for any formula χ of RPTL of lower complexity than ϕ where $\mathcal{N}$ is defined below):

- (Base case: $\phi = p$) $\mathcal{A}(p) = (\{s\}, s, \delta, \Omega)$ where $\delta(s) = p, \Omega(s) = 1$.
- (Inductive case: $\phi = \neg\psi$) $\mathcal{A}(\phi) = \mathcal{N}(\psi)$ where $\mathcal{N}(\psi)$ is defined below.
- (Inductive case: $\phi = \psi_1 \wedge \psi_2$) $\mathcal{A}(\phi) = \mathcal{A}(\psi_1) \cdot \mathcal{A}(\psi_2)$.
- (Inductive case: $\phi = [\pi]\psi_1\mathcal{U}\psi_2\mathcal{T}\psi_3$)

By induction that we already have $\mathcal{A}(\psi_1) = (S^{\psi_1}, s_I^{\psi_1}, \delta^{\psi_1}, \Omega^{\psi_1})$, $\mathcal{A}(\psi_2) = (S^{\psi_2}, s_I^{\psi_2}, \delta^{\psi_2}, \Omega^{\psi_2})$, and $\mathcal{A}(\psi_3) = (S^{\psi_3}, s_I^{\psi_3}, \delta^{\psi_3}, \Omega^{\psi_3})$, and by assumption we have $\mathcal{N}(\psi_1) = (S^{\neg\psi_1}, s_I^{\neg\psi_1}, \delta^{\neg\psi_1}, \Omega^{\neg\psi_1})$, $\mathcal{N}(\psi_2) = (S^{\neg\psi_2}, s_I^{\neg\psi_2}, \delta^{\neg\psi_2}, \Omega^{\neg\psi_2})$.

But as we saw above we can also define the ATA $\mathcal{D}[\pi]p\mathcal{U}q\mathcal{T}r = (S', s'_I, s'_p, s'_{\neg p}, s'_q, s'_{\neg q}, s'_f, \delta', \Omega')$, where p, q and r can be any triple of distinct propositional variables not in ϕ (it doesn't matter which distinct triple we choose the resulting DATAs will be isomorphic to each other). We assume that $S^{\psi_i}, S^{\neg\psi_i}$, and S' are disjoint.

Then $\mathcal{A}([\pi]\psi_1\mathcal{U}\psi_2\mathcal{T}\psi_3) = (S^{\phi_1} \cup S^{\phi_2} \cup S^{\phi_3} \cup S^{\neg\phi_1} \cup S^{\neg\phi_2} \cup S', s'_I, \delta, \Omega)$ where $\Omega(s) = \Omega^{\phi_i}(s)$ for $s \in S^{\phi_i}$, $\Omega(s) = \Omega^{\neg\phi_i}(s)$ for $s \in S^{\neg\phi_i}$ and $\Omega(s) = \Omega'(s)$ for $s \in S'$. The transition function δ is defined as follows:

$$\delta(s) = \begin{cases} \delta^{\psi_1}(s) & \text{if } s \in S^{\psi_1}, \\ \delta^{\psi_2}(s) & \text{if } s \in S^{\psi_2}, \\ \delta^{\psi_3}(s) & \text{if } s \in S^{\psi_3}, \\ \delta^{\neg\psi_1}(s) & \text{if } s \in S^{\neg\psi_1}, \\ \delta^{\neg\psi_2}(s) & \text{if } s \in S^{\neg\psi_2}, \\ \delta'(s) & \text{if } s \in S' - \{s'_p, s'_{\neg p}, s'_q, s'_{\neg q}, s'_f\}, \\ s_I^{\psi_1} & \text{if } s = s'_p, \\ s_I^{\neg\psi_1} & \text{if } s = s'_{\neg p}, \\ s_I^{\psi_2} & \text{if } s = s'_q, \\ s_I^{\neg\psi_2} & \text{if } s = s'_{\neg q}, \\ s_I^{\psi_3} & \text{if } s = s'_f. \end{cases}$$

We define $\mathcal{N}(\phi)$ inductively as follows for any RPTL formula ϕ (for this definition we assume that we can define $\mathcal{A}(\chi)$ for any formula χ of RPTL of lower complexity than ϕ):

- (Base case: $\phi = p$) $\mathcal{N}(p) = (\{s\}, s, \delta, \Omega)$ where $\delta(s) = \neg p, \Omega(s) = 1$.
- (Inductive case: $\phi = \neg\psi$) $\mathcal{N}(\phi) = \mathcal{A}(\psi)$.
- (Inductive case: $\phi = \psi_1 \wedge \psi_2$) $\mathcal{N}(\phi) = \mathcal{N}(\psi_1) + \mathcal{N}(\psi_2)$.
- (Inductive case: $\phi = [\pi]\psi_1\mathcal{U}\psi_2\mathcal{T}\psi_3$)

By induction that we already have $\mathcal{A}(\psi_1) = (S^{\psi_1}, s_I^{\psi_1}, \delta^{\psi_1}, \Omega^{\psi_1})$, $\mathcal{A}(\psi_2) = (S^{\psi_2}, s_I^{\psi_2}, \delta^{\psi_2}, \Omega^{\psi_2})$, and by assumption we have $\mathcal{N}(\psi_1) = (S^{\neg\psi_1}, s_I^{\neg\psi_1}, \delta^{\neg\psi_1}, \Omega^{\neg\psi_1})$, $\mathcal{N}(\psi_2) = (S^{\neg\psi_2}, s_I^{\neg\psi_2}, \delta^{\neg\psi_2}, \Omega^{\neg\psi_2})$ and $\mathcal{N}(\psi_3) = (S^{\neg\psi_3}, s_I^{\neg\psi_3}, \delta^{\neg\psi_3}, \Omega^{\neg\psi_3})$. Then we define $\mathcal{N}^{\psi_1 \vee \psi_2} = \mathcal{N}(\psi_1) \cdot \mathcal{N}(\psi_2) = (S^\wedge, s_I^\wedge, \delta^\wedge, \Omega^\wedge)$ and $\mathcal{A}^{\psi_1 \vee \psi_2} = \mathcal{A}(\psi_1) + \mathcal{A}(\psi_2) = (S^\vee, s_I^\vee, \delta^\vee, \Omega^\vee)$.

But as we saw above we can also define the ATA $\mathcal{D}[\pi]p\mathcal{U}q\mathcal{T}r = (S', s'_I, s'_p, s'_{\neg p}, s'_q, s'_{\neg q}, s'_f, \delta', \Omega')$, where p, q and r can be any triple of distinct propositional variables not in ϕ (it doesn't matter which distinct triple we choose the resulting DATAs will be isomorphic to each other). We assume that $S^{\psi_i}, S^{\neg\psi_i}, S^\wedge, S^\vee$ and S' are disjoint.

Then $\mathcal{A}([\pi]\psi_1\mathcal{U}\psi_2\mathcal{T}\psi_3) = (S^{\phi_1} \cup S^{\phi_2} \cup S^\vee \cup S^\wedge \cup S^{\neg\phi_2} \cup S^{\neg\phi_3} \cup S', s'_I, \delta, \Omega)$ where $\Omega(s) = \Omega^{\phi_i}(s)$ for $s \in S^{\phi_i}$, $\Omega(s) = \Omega^{\neg\phi_i}(s)$ for $s \in S^{\neg\phi_i}$, $\Omega(s) = \Omega^\wedge(s)$ for $s \in S^\wedge$, $\Omega(s) = \Omega^\vee(s)$ for $s \in S^\vee$ and $\Omega(s) = \Omega'(s)$ for $s \in S'$. The transition function δ is defined as follows:

$$\delta(s) = \begin{cases} \delta^{\neg\psi_2}(s) & \text{if } s \in S^{\neg\psi_2}, \\ \delta^{\psi_2}(s) & \text{if } s \in S^{\psi_2}, \\ \delta^\wedge(s) & \text{if } s \in S^\wedge, \\ \delta^\vee(s) & \text{if } s \in S^\vee, \\ \delta^{\neg\psi_3}(s) & \text{if } s \in S^{\neg\psi_3}, \\ \delta'(s) & \text{if } s \in S' - \{s'_p, s'_{\neg p}, s'_q, s'_{\neg q}, s'_f\}, \\ s_I^{\neg\psi_2} & \text{if } s = s'_p, \\ s_I^{\psi_2} & \text{if } s = s'_{\neg p}, \\ s_I^\wedge & \text{if } s = s'_q, \\ s_I^\vee & \text{if } s = s'_{\neg q}, \\ s_I^{\neg\psi_3} & \text{if } s = s'_f. \end{cases}$$

Theorem 4 For any pointed Kripke structure $(\mathcal{M}, w)$ where $\mathcal{M} = (W, \tau, \kappa)$ and RPTL formula $\phi, (\mathcal{M}, w) \models \phi$ if and only if $(\mathcal{M}, w) \in \|\mathcal{A}(\phi)\|$

The proof is by structural induction on formulae of ϕ , we can prove the following subresult.

Theorem 5 For any pointed Kripke structure $(\mathcal{M}, w)$ where $\mathcal{M} = (W, \tau, \kappa)$ and RPTL formula $\neg\phi$. Given that the Theorem 4 holds for all formulae of lower complexity than ϕ we have, $(\mathcal{M}, w) \models \neg\phi$ if and only if $(\mathcal{M}, w) \in \|\mathcal{N}(\phi)\|$

This clearly gives us an ATA that is polynomial in the size of our original formula and since the nonemptiness problem for ATA is in EXPTIME, gives us an EXPTIME upper-bound.

2.6 Conclusion

In this paper we have introduced a new logic RPTL and proven its decidability and in so doing have established the complexity of its satisfiability problem. However we have not yet established whether or not RPTL is actually equivalent in expressive power to PDL, i.e., whether or not RPTL can be translated into PDL. It also remains to be seen whether an axiomatisation can be found. We have also yet to explore model checking issues for RPTL.

REFERENCES

- [1] Natasha Alechina, Mehdi Dastani, Brian Logan, and John-Jules Ch. Meyer, 'A logic of agent programs', in AAI, pp. 795–800, (2007).
- [2] Ilan Beer, Shoham Ben-David, and Avner Landver, 'On-the-fly model checking of rctl formulas', in CAV '98: Proceedings of the 10th International Conference on Computer Aided Verification, pp. 184–194, London, UK, (1998). Springer-Verlag.
- [3] Jens Claßen and Gerhard Lakemeyer, 'A logic for non-terminating golog programs', in Principles of Knowledge Representation and Reasoning: Proceedings of the Eleventh International Conference, KR 2008, Sydney, Australia, September 16–19, 2008, eds., Gerhard Brewka and Jérôme Lang, pp. 589–599. AAAI Press, (2008).
- [4] Mads Dam, 'Ctl* and ectl* as fragments of the modal mu-calculus', Theor. Comput. Sci., **126**(1), 77–96, (1994).
- [5] David Harel and David Peleg, 'Process logic with regular formulas', Theor. Comput. Sci., **38**, 307–322, (1985).
- [6] M Y Vardi and P Wolper, 'Yet another process logic', in Logics of Programs LNCS 164, pp. 501–512, (1983).
- [7] Moshe Y. Vardi, 'Alternating automata: Checking truth and validity for temporal logics', in Proceedings of the 14th International Conference on Automated Deduction (CADE'97), ed., William McCune, volume 1249 of Lecture Notes in Artificial Intelligence, pp. 191–206. Springer-Verlag, (July 1997).
- [8] Thomas Wilke, 'Alternating tree automata, parity games, and modal μ -calculus', Bull. Soc. Math. Belg., **8**(2), (May 2001).