On the Expressivity of RoCTL*

John M[c]Cabe-Dansted*, Tim French, Mark Reynolds
The University of Western Australia
Computer Science and Software Engineering
{john,tim,mark}@csse.uwa.edu.au

Sophie Pinchinat
Campus Universitaire de Beaulieu
IRISA
Sophie.Pinchinat@irisa.fr

Abstract

RoCTL was proposed to model robustness in concurrent systems. RoCTL* extended CTL* with the addition of Obligatory and Robustly operators, which quantify over failure-free paths and paths with one more failure respectively. Whether RoCTL* is more expressive than CTL* has remained an open problem since the RoCTL* logic was proposed. We use the equivalence of LTL to counter-free automata to show that RoCTL* is expressively equivalent to CTL*; the translation to CTL* provides the first model checking procedure for RoCTL*. However, we show that RoCTL* is relatively succinct as all satisfaction preserving translations into CTL* are non-elementary in length.* ***Draft: May 19, 2009***

1. Introduction

The Robust Full Computation Tree Logic (RoCTL*) [11] is an extension of CTL* introduced to represent issues relating to robustness and reliability in systems. It does this by adding an Obligatory operator and a Robustly operator. The Obligatory operator specifies how the systems should behave by quantifying over paths in which no failures occur. The Robustly operator specifies that something must be true on the current path and similar paths that "deviate" from the current path, having at most one more failure occurring. This notation allows phrases such as "even with n additional failures" to be built up by chaining n simple unary Robustly operators together.

The RoCTL* Obligatory operator is similar to the Obligatory operator in Standard Deontic Logic (SDL), although in RoCTL* the operator quantifies over paths rather than worlds. SDL has many paradoxes. Some of these, such as the "Gentle Murderer" paradox spring from the inadequacy of SDL for dealing with obligations caused by acting contrary to duty such as "If you murder, you must murder gently". Contrary-to-Duty (CtD) obligations are important for modeling a robust system, as it is often important to state that the system should achieve some goal and also that if it fails it should in some way recover from the failure. RoCTL* can represent CtD obligations by specifying that the agent must ensure that the CtD obligation is met even if a failure occurs. For further discussion of CtD obligations and motivations for RoCTL*, see [11]. The obligatory operator, as well as some uses of the robustly operator, are easy to translate into CTL* [17].

When RoCTL* was originally proposed [11], it had two accessibility relations, a success and failure transition. However we may equivalently define RoCTL* with a single accessibility relation if we add a violation atom $\mathbf{v}$ to indicate that the previous transition was a failure transition (this new definition was first used in [17]). Under this definition, the RoCTL* models are CTL models, albeit with a special violation atom not used in RoCTL* formulas. In this paper we show that if we allow the violation atom to occur in CTL* formulas, then we can express every RoCTL* formula into an equivalent CTL* formula. Although we will expose the violation atom, we do not extend the model, in

*This Project is supported by the Australian Government's International Science Linkages program

particular we do not add atoms to the model.

The addition of the Robustly operator and temporal operators to Deontic logic allows RoCTL* to deal with Contrary-to-Duty obligations. SDL is able to distinguish what ought to be true from what is true, but is unable to specify obligations that come into force only when we behave incorrectly. For example, SDL is inadequate to represent the obligation "if you murder, you must murder gently" [10]. Addition of temporal operators to Deontic logic allows us to specify correct responses to failures that have occurred in the past [23]. However, this approach alone is not sufficient [23] to represent obligations such as "You must assist your neighbour, and you must warn them iff you will not assist them". In RoCTL* these obligations can be represented if the obligation to warn your neighbour is robust but the obligation to assist them is not.

Other approaches to dealing with Contrary-to-Duty obligations exist. Defeasible logic is often used [18], and logics of agency, such as STIT [3], can be useful as they can allow obligations to be conditional on the agent's ability to carry out the obligation.

This paper provides some examples of robust systems that can be effectively represented in RoCTL*. It is easy to solve the coordinated attack problem if our protocol is allowed to assume that only n messages will be lost. The logic may also be useful to represent the resilience of some economy to temporary failures to acquire or send some resource. For example, a remote mining colony may have interacting requirements for communications, food, electricity and fuel. RoCTL* may be more suitable than Resource Logics (see e.g. [6]) for representing systems where a failure may cause a resource to become temporarily unavailable. This paper presents a simple example where the only requirement is to provide a cat with food when it is hungry.

A number of other extensions of temporal logics have been proposed to deal with Deontic or Robustness issues [4, 16, 13, 1, 21]. Each of these logics are substantially different from RoCTL*. Some of these logics are designed specifically to deal with deadlines [4, 13]. An Agent Communication Language was formed by adding Deontic and other modal operators to CTL [21]; this language does not explicitly deal with robustness or failures. Hansson and Johnsson [13] proposed an extension of CTL to deal with reliability. However their logic reasons about reliability using probabilities rather than numbers of failures, and their paper does not contain any discussion of the relationship of their logic to Deontic logics. Like our embedding into QCTL*, Aldewereld et al. [1] uses a Viol atom to represent failure. However, their logic also uses probability instead of failure counts and is thus suited to a different class of problems than RoCTL*. None of these logics appear to have an operator that is substantially similar to the Robustly operator of RoCTL*.

Diagnosis problems in control theory [14, 2] also deals with failures of systems. Diagnosis is in some sense the dual of the purpose of the RoCTL* logic, as diagnosis requires that failure cause something (detection of the failure) whereas robustness involves showing that failure will *not* cause something.

The translation we will present in this paper results in a formula that is satisfied on a model iff the original formula is satisfied on the same model. This means that we can use all the CTL* model checkers, decision procedures and so forth for RoCTL*.

We will then show that although all RoCTL* formulas can be translated into CTL*, the length of the CTL* formula is not elementary in the length of the RoCTL* formula. Hence some properties can be represented much more succinctly in RoCTL* than CTL*.

2. Definitions

2.1. RoCTL-Structures and Trees

Definition 1. We let $\mathcal{V}$ be our set of variables. The set $\mathcal{V}$ contains a special variable $\mathbf{v}$. A *valuation* g is a map from a set of worlds A to the power set of the variables. The statement $p \in g(w)$ means roughly "the variable p is true at world w".

Definition 2. We say that a binary relation R on S is *serial* (total) if for every a in S there exists b in S such that aRb.

Definition 3. A *structure* $M = (A, R, g)$ is a 3-tuple containing a set of worlds A, a serial binary relation R on A, a valuation g on the set of worlds A.

Definition 4. We call an ω-sequence $\sigma = \langle w_0, w_1, \ldots \rangle$ of worlds a *fullpath* iff for all non-negative integers i we have $w_i R w_{i+1}$. For all i in $\mathbb{N}$ we define $\sigma_{\geq i}$ to be the fullpath $\langle w_i, w_{i+1}, \ldots \rangle$, we define σ_i to be w_i and we define $\sigma_{\leq i}$ to be the sequence $\langle w_0, w_1, \ldots, w_i \rangle$.

Definition 5. We say that a fullpath σ is *failure-free* iff for all $i > 0$ we have $\mathbf{v} \notin g(\sigma_i)$. We define $SF(w)$ to be the set of all fullpaths starting with world w and $S(w)$ to be the set of all failure-free fullpaths starting with w. We call a CTL structure a RoCTL structure iff $S(w)$ is non-empty for every $w \in A$.

Definition 6. For two fullpaths σ and π we say that π is an *i-deviation* from σ iff $\sigma_{\leq i} = \pi_{\leq i}$ and $\pi_{\geq i+1} \in S(\pi_{i+1})$. We say that π is a *deviation* from σ if there exists a non-negative integer i such that π is an i-deviation from σ. We define a function δ from fullpaths to sets of fullpaths such that where σ and π are fullpaths, π is a member of $\delta(\sigma)$ iff π is a deviation from σ.

We see that $S(\sigma_0) \subseteq \delta(\sigma) \subseteq SF(\sigma_0)$. Where p varies over $\mathcal{V}$, we define RoCTL* formulas according to the following abstract syntax

$$\phi := \top \mid p \mid \neg\phi \mid (\phi \wedge \phi) \mid (\phi U \phi) \mid N\phi \mid A\phi \mid O\phi \mid \blacktriangle\phi \,.$$

A formula that begins with A, $\neg A$, O, $\neg O$, p or $\neg p$ is called a *state formula*. For consistency with [11], we do not consider a formula that explicitly contains $\mathbf{v}$ to be a RoCTL* formula, although our translation into CTL* works equally well for such formulas. The $\top$, $\neg$, $\wedge$, N, U and A are the familiar "true", "not", "and", "next", "until" and "all paths" operators from CTL. The abbreviations $\bot$, $\vee$, F, G, W, E $\rightarrow$ and $\leftrightarrow$ are defined as in CTL* logic. As with Standard Deontic Logic (SDL) logic, we define $P \equiv \neg O \neg$. Finally, we define the dual $\triangle$ of $\blacktriangle$ as the abbreviation $\Delta \equiv \neg\blacktriangle\neg$. We call the O, P, $\blacktriangle$, Δ operators Obligatory, Permissible, Robustly and Prone respectively.

We define truth of a RoCTL* formula ϕ on a fullpath $\sigma = \langle w_0, w_1, \ldots \rangle$ in a RoCTL-structure M recursively as follows:

$$\begin{aligned}
&M, \sigma \vDash N\phi \text{ iff } M, \sigma_{\geq 1} \vDash \phi \\
&M, \sigma \vDash \phi U \psi \text{ iff } \exists_{i \in \mathbb{N}} \text{ s.t. } M, \sigma_{\geq i} \vDash \psi \text{ and} \\
&\qquad\qquad \forall_{j \in \mathbb{N}} j < i \implies M, \sigma_{\geq j} \vDash \phi \\
&M, \sigma \vDash A\phi \text{ iff } \forall_{\pi \in SF(\sigma_0)} M, \pi \vDash \phi \\
&M, \sigma \vDash O\phi \text{ iff } \forall_{\pi \in S(\sigma_0)} M, \pi \vDash \phi \\
&M, \sigma \vDash \blacktriangle\phi \text{ iff } \forall_{\pi \in \delta(\sigma)} M, \pi \vDash \phi \text{ and } M, \sigma \vDash \phi
\end{aligned}$$

The definition for $\top$, p, $\neg$ and $\wedge$ is as we would expect from classical logic. The intuition behind the $\blacktriangle$ operator is that it quantifies over paths that could result if a single error was introduced; the deviations only have at most one failure not on the original path, and they are identical to the original path until this failure occurs.

Definition 7. We say that a function τ from formulas to formulas is *satisfaction preserving* iff for all M, σ and ϕ it is the case that $M, \sigma \vDash \phi \iff M, \sigma \vDash \tau(\phi)$.

We will now define a tree. A tree is similar to a structure, but a tree need not be serial, and each node only has one parent.

Definition 8. We say $T = (A, R, g)$ is a $\mathcal{V}$-labelled *tree*, for some set $\mathcal{V}$, iff

1. A is a non-empty set of nodes
2. for all $x, y, z \in A$ if $(x, z) \in R$ and $(y, z) \in R$ then $x = y$.
3. there does not exist any cycle $x_0 R \cdots R x_0$ through R.
4. there exists a node x such that for all $y \in A$, if $y \neq x$ there exists a sequence $xRx_1R \cdots Ry$ through R.
5. the valuation g (or labeling) is a function from A to $2^{\mathcal{V}}$, that is for each $w \in A$, $g(w) \subseteq \mathcal{V}$.

Definition 9. We define the height of a finite tree $T = (A, R, g)$ as follows: **root** is a function from trees to nodes such that $\mathbf{root}(T)$ is the root of the tree T. $\mathbf{height}(T) = \mathbf{height}_R(\mathbf{root}(T))$ where $\mathbf{height}_R$ is a function from A to $\mathbb{N}$ such that for all $x \in A$, we let $\mathbf{height}_R(x)$ be the smallest non-negative integer such that $\mathbf{height}_R(x) > \mathbf{height}_R(y)$ for all y such that $(x, y) \in R$.

Definition 10. We say $C = \langle A_C, R_C, g_C \rangle$ is a *subtree* of $T = (A, R, g)$ iff there exists $w \in A$ such that A_C is the subset of A reachable from w and R_C and g_C are the fragments of R and g on A_C respectively. We say C is a *direct subtree* of $T = (A, R, g)$ if C is a subtree of T and $(\mathbf{root}(T), \mathbf{root}(C)) \in R$.

2.2. Automata

Definition 11. A Büchi automaton $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ contains

Σ: set of symbols (alphabet)
S: finite set of automaton states
S_0: set of initial states $\subseteq S$
δ : a transition function $\subseteq (S \times \Sigma \times S)$
F: A set of accepting states $\subseteq 2^S$

We call the members of Σ^* words. Unlike a path through a structure, each transition of a path through an automaton is labelled with an element e of Σ. We say $s_0 \overset{e_0}{\rightarrow} s_1 \overset{e_1}{\rightarrow} \cdots \overset{e_{n-1}}{\rightarrow} s_n$ is a path of $\mathcal{A}$ if for all $0 \leq i < n$ the tuple $\langle s_i, e_i, s_{i+1} \rangle$ is in δ. The label of the path is $\langle e_0, e_1, \ldots, e_n \rangle$. Let $L_{p,q}(\mathcal{A})$ be the set of all labels of paths through $\mathcal{A}$ from p to q.

A run ρ of $\mathcal{A}$ is a path starting at a state in S_0. We say an infinite run is accepting if a state in F occurs infinitely often in the run.

In this paper, $\Sigma = 2^{\Phi}$, for some set of state formulas Φ. Given a fixed structure $M = (A, \rightarrow, g)$ and path σ, we let $g_\Phi(\sigma_i) = \{\phi : \phi \in \Phi \wedge M, \sigma_{\geq i} \vDash \phi\}$ and $g_\Phi(\sigma) = \langle g_\Phi(\sigma_0), g_\Phi(\sigma_1), \ldots \rangle$. Note that Φ consists solely of state formulas so $g_\Phi(\sigma_0) = g_\Phi(\pi_0)$ if $\sigma_0 = \pi_0$.

Definition 12. A *counter-free* automaton is an automaton such that for all positive integers m, states $s \in S$ and words u in Σ^*, if $u^m \in L_{s,s}$ then $u \in L_{s,s}$ [7].

Above we have defined linear automata. These are sufficient for the proof of expressive equivalence of RoCTL* and CTL*. However, in the proof that RoCTL* is relatively succinct, we will use tree automata. We will define a type of tree automata called a symmetric alternating automata (SAA) (see e.g. [15]), these are a subclass of alternating automata, and can also be referred to as just alternating automata (see e.g. [24]).

Every node, in the run of an SAA on an input structure M, represents a world of M. However, a world w in the input structure M may occur many times in a run. Where a non-deterministic automata would non-deterministically pick a next state, a SAA non-deterministically picks a conjunction of elements of the form $(\Box, q)$ and $(\Diamond, q)$; alternatively we may define SAA as deterministically picking a Boolean combination of requirements of this form, see for example [15]. Alternating automata can also be thought of as a type of parity game, see for example [12]. An element of the form $(\Box, q)$/$(\Diamond, q)$ indicates for every/some child u of the current world w of the input structure M, a run on M must have a branch which follows u and where q is the next state.

Definition 13. A parity acceptance condition F of an automata $(\Sigma, S, S_0, \delta, F)$ is a map from S to $\mathbb{N}$. We say that parity condition accepts an infinite path if the largest integer n, such that $F(q) = n$ for some q that occurs infinitely often on the path, is even.

Definition 14. A *symmetric alternating automata* (SAA) is a tuple $(\Sigma, S, S_0, \delta, F)$ where Σ, S and S_0 are defined as in Büchi automata, and

δ : a transition function $\subseteq (S \times \Sigma \times 2^{\{\Box, \Diamond\} \times S})$

We define the the acceptance condition F of an SAA to be a parity acceptance condition, but note that we can express Büchi parity conditions as parity acceptance conditions. The SAA accepts a run iff every infinite path through the run satisfies F.

A run $R = \langle A_R, R_R, g_R \rangle$ of the SAA on a $\mathcal{V}$-labelled input structure M is an $A \times S$-labelled tree structure. Where $g_R(\mathbf{root}(R)) = (w, q)$, it is the case that $q \in S$ and $w = \mathbf{root}(M)$. For every w_R in A_r, where $(w, q) = g_R(w_R)$ and $e = g(w)$, there exists some set $X \in 2^{\{\Box, \Diamond\} \times S}$ such that $(w_R, e, X) \in \delta$ and

1. For each $r \in S$ such that $(\Box, r) \in X$, for *each* u such that wRu there must exist u_R such that $w_R R u_R$ and $(u, r) \in g_R(u_R)$.

2. For each $r \in S$ such that $(\Diamond, r) \in X$, for *some* u such that wRu there must exist u_R such that $w_R R u_R$ and $(u, r) \in g_R(u_R)$.

3. Examples

In this section a number of examples are presented. The first example examines the difference between the formula $NO\phi$ and the formula $ON\phi$. The second example shows how RoCTL* may be used to specify a robust network protocol. Then an example of feeding a cat will be introduced to explain how we may reason about consequences of polices. These examples will frequently use the ▲/Δ operator to form the pair O▲. In the final example we use the simple formula $O(\triangle Fe \rightarrow Fw)$ which nests ▲/Δ in a less trivial way.

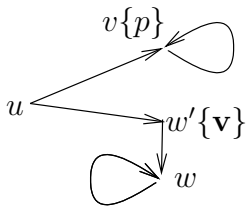
Example 15. Here is an example of a simple Contrary-to-Duty obligation. This provides a counter example to both $ON\phi \rightarrow NO\phi$ and $NO\phi \rightarrow ON\phi$.

$ON(Gp)$**:** You should commit to the proper decision.

$NO\left(G\neg p \vee Gp\right)$**:** Once you have made your decision, you must stick with it.

It is consistent with the above that we do not make the proper decision ($N\neg p$). Once we have made the wrong decision we cannot satisfy Gp, so we must stick with the wrong decision $G\neg p$. Hence, in this case, both $ON(Gp)$ and $NO(G\neg p)$ are true. Likewise $ON(G\neg p)$ and $NO(Gp)$ are false. This demonstrates how obligations can change with time in RoCTL*. We will now give an example of a structure $M = (A, R, g)$ that satisfies these formulæ:

$$\begin{aligned} A &= \{u, v, w, w'\}, \\ R &= \{(u,v),(v,v),(u,w'),(w',w)(w,w)\}, \\ g(v) &= \{p\}, \quad g(w) = g(u) = \emptyset, \quad g(w') = \{\mathbf{v}\}\ . \end{aligned}$$



Let σ be the fullpath $\langle u, w, w, \ldots\rangle$ corresponding to making the wrong decision. We see that $M, \sigma_{\geq 1} \vDash \neg p$, so $M, \sigma_{\geq 1} \vDash O\neg p$ and $M, \sigma_{\geq 1} \vDash \neg Op$. Thus $M, \sigma \vDash NO\neg p$ and $M, \sigma \vDash N\neg Op$. It follows that $M, \sigma \vDash \neg NOp$.

Let $\pi = \langle v, v, \ldots\rangle$. We see that $M, \pi \vDash p$. We see that $S(u) = \{\langle u, v, v, \ldots\rangle\}$. Hence $M, \sigma \vDash ONp$ and it follows that $M, \sigma \vDash \neg O\neg Np$ and so $M, \sigma \vDash \neg ON\neg p$.

Hence $M, \sigma \vDash (ONp \wedge \neg NOp)$ and so $M, \sigma \nvDash (ON\phi \rightarrow NO\phi)$ where $\phi = p$. Likewise $M, \sigma \vDash (NO\neg p \wedge \neg ON\neg p)$, so $M, \sigma \nvDash (NO\phi \rightarrow ON\phi)$ where $\phi = \neg p$.

Example 16. In the coordinated attack problem we have two generals X and Y. General X wants to organise an attack with Y. A communication protocol will be presented such that a coordinated attack will occur if no more than one message is lost.

$AG\left(s_X \rightarrow ONr_Y\right)$**:** If X sends a message, Y should receive it at the next step.

$AG\left(\neg s_X \rightarrow \neg Nr_Y\right)$**:** If X does not send a message now, Y will not receive a message at the next step.

$AG(f_X \rightarrow AGf_X)$**:** If X commits to an attack, X cannot withdraw.

$AG(f_X \rightarrow \neg s_X)$**:** If X has committed to an attack, it is too late to send messages.

$A\left(\neg f_X W r_X\right)$**:** X cannot commit to an attack until X has received plans from Y

Similar constraints to the above also apply to Y. Below we add a constraints requiring X to be the general planning the attack

$A\left(\neg s_Y W r_Y\right)$**:** General Y will not send a message until Y has received a message.

No protocol exists to satisfy the original coordination problem, since an unbounded number of messages can be lost. Here we only attempt to ensure correct behaviour if one or fewer messages are lost.

$A\left(s_X U r_X\right)$**:** General X will send plans until a response is received.

$AG\left(r_X \rightarrow f_X\right)$**:** Once general X receives a response, X will commit to an attack.

$A\left(\neg r_Y W\left(r_Y \wedge \left(s_Y \wedge Ns_Y \wedge NNf_Y\right)\right)\right)$**:** Once general Y receives plans, Y will send two messages to X and then commit to an attack.

Having the formal statement of the policy above and the semantics of RoCTL* we may prove that the policy $\hat{\phi}$ is consistent and that it implies correct behaviour even if a single failure occurs:

$$\hat{\phi} \to O\blacktriangle F(f_X \wedge f_Y)\,.$$

Indeed, we have shown that such issues can be decided in finite time [19] .

For a more thorough specification of the Coordinated Attack problem, see for example [20, 22]. .

Example 17. We have a cat that does not eat the hour after it has eaten. If the cat bowl is empty we might forget to fill it. We must ensure that the cat never goes hungry, even if we forget to fill the cat bowl one hour. At the beginning of the first hour, the cat bowl is full. We have the following variables:

b "The cat bowl is full at the beginning of this hour"

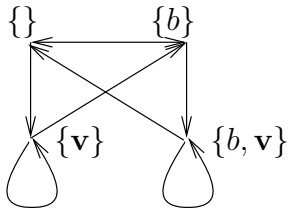
d "This hour is feeding time"

We can translate the statements above into RoCTL* statements:

1. $AG(d \to \neg Nd)$: If this hour is feeding time, the next is not.
2. $AG((d \vee \neg b) \to \Delta N\neg b)$: If it is feeding time or the cat bowl was empty, a single failure may result in an empty bowl at the next step
3. $AG((\neg d \wedge b) \to Nb)$: If the bowl is full and it is not feeding time, the bowl will be full at the beginning of the next hour.
4. $O\blacktriangle G\,(d \to b)$: It is obligatory that, even if a single failure occurs, it is always the case that the bowl must be full at feeding time.
5. b: The cat bowl starts full.

Having the formalised the policy it can be proven that the policy is consistent and that the policy implies $O\blacktriangle GONb$, indicating that the bowl must be filled at every step (in case we forget at the next step), unless we have already failed twice. The formula $AGONb \to O\blacktriangle G\,(d \to b)$ can also be derived, indicating that following a policy requiring us to always attempt to fill the cat bowl ensures that we will not starve the cat even if we make a single mistake. Thus following this simpler policy is sufficient to discharge our original obligation.

Example 18. Say that a bit ought to flip at every step, but might fail to flip at any particular step. This may be represented with the RoCTL* statement $AGO\,(b \leftrightarrow \neg Nb) \wedge AG\triangle\,(b \leftrightarrow Nb)$, which is satisfied by the following model:



Then we may derive the following statements:

$O\blacktriangle((b \wedge Nb) \to NG\,(b \leftrightarrow \neg Nb))$ If a single failure occurs, and the bit fails to flip at the next step, it will flip continuously from then on.

$O\blacktriangle FG\,(b \leftrightarrow \neg Nb)$ Even if a single failure occurs, there will be time at which the bit will flip correctly from then on.

However, we will not be able to derive $OF\blacktriangle G\,(b \leftrightarrow \neg Nb)$, as this would mean that there was a time at which a failure could not cause the bit to miss a step.

Example 19. We define a system that will warn the user if the system enters an unsafe state:

1. $AGONs$: The system should always ensure that the system reaches a safe state by the next step.
2. $AG(s \to N\neg e)$: If the system is in a safe state an error e will not occur at the next step.
3. $s \wedge \neg e$: The system starts in a safe state with no error.
4. $AG(\neg s \to Nw)$: If the system is in an unsafe state, the system will warn the user at the next step.

We may prove that if an error e almost occurs, the system will finally warn the user, i.e. $O(\triangle Fe \to Fw)$.

Example 20. Say that we have wireless sensor and a base station. Upon detecting some event, the wireless sensor will activate and send three packets to the base station. The base station will not know that the wireless sensor sent data if all three packets were lost. Thus an error will be reported iff the base station receives either one or two packets. This can be formalised as

$s \wedge Ns \wedge NNs \wedge NNNG\neg s$: The sensor will send three packets.

$AG\left(s \rightarrow ONr \wedge \neg s \rightarrow N\neg s\right)$: If a packet is sent, it should be received at the next step. If it is not sent it will not be received.

$\neg N\left(\left(r \wedge Nr \wedge NNr\right) \vee G\neg r\right) \rightarrow NNNe$: An error will be detected if some packets, but not all three, are received.

It follows that $O\blacktriangle\left(\triangle FeU\neg s\right)$, indicating that it is robustly true that if an additional failure occurs, an error could be detected. In this example a failure may not indicate a packet being dropped, e.g. it has not been specified whether the packet arrives corrupted. Thus the system cannot detect all failures. In RoCTL* it is impossible to specify that a failure will have an effect. At best we can specify that it is always *possible* for a failure to be detected. However, we can specify that some particular effect will be detected. For example, we can express the statement "Even if two or fewer packets are lost, either all packets arrive or an error is detected" as

$$O\blacktriangle\blacktriangle N\left(\left(r \wedge Nr \wedge NNr\right) \vee Fe\right) .$$

Example 21. Say a system has a battery that can sustain the system for a single step, even if a failure occurs (the fuse blows). Let ϕ represent "the system has power now and at the next step". Then, even if a single failure occurs, it will always be the case that even if a deviating event occurs the system will have power now and at the next step ($OG\blacktriangle\phi$). It would not follow that even if a single failure occurred the system would always have power ($O\blacktriangle G\phi$); the battery power would only last one step after the fuse blew. If we also specified that the fuse was an electronic fuse that automatically reset, then if a single failure occurs, the system would only have to rely on battery power for one step. Then, if the fuse only blows once then system will always have power ($\blacktriangle G\phi$). As with the A operator in CTL*, $\blacktriangle G\phi \rightarrow G\blacktriangle\phi$ is valid in RoCTL* but $G\blacktriangle\phi \rightarrow \blacktriangle G\phi$ is not.

4. Expressivity

We will define a translation of $\triangle\phi$ into CTL*, for any CTL* formula ϕ. We will first translate ϕ into counter-free Büchi automata $\mathcal{A}$, we will then define a function $\tau_{\triangle}$ from automata to automata such $\tau_{\triangle}\left(\mathcal{A}\right)$ is equivalent to $\triangle\phi$, and translate $\tau_{\triangle}\left(\mathcal{A}\right)$ into CTL*. This allows us to recursively translate any RoCTL* formula into an equivalent CTL* formula.

4.1 CTL* and LTL

Theorem 22. *A language L is definable in LTL iff L is accepted by some counter-free Büchi automaton [7].*

It is well known that we can express a CTL* formula as an LTL formula over a path, where that path includes state formula as atoms; this is commonly used in model checking [8, 9, 5]. From the above theorem we can also express this LTL formula as a Büchi automaton.

Formally, for any CTL* formula ϕ there exists a set of state formulas Φ and a counter-free automaton $\mathcal{A} = \left(2^{\Phi}, S, S_0, \delta, F\right)$ such that $\mathcal{A}$ accepts $g_{\Phi}\left(\sigma\right)$ iff $M, \sigma \vDash \phi$. We say an automaton $\mathcal{A} = \left(2^{\Phi}, S, S_0, \delta, F\right)$ is equivalent to a formula ϕ iff

$$\left(\forall_{M,\sigma} M, \sigma \vDash \phi\right) \iff \left(\mathcal{A} \text{ accepts } g_{\Phi}\left(\sigma\right)\right) .$$

Definition 23. Let $\mathfrak{A}$ be a function from CTL* formulas to counter-free Büchi automata such that $\mathfrak{A}\left(\phi\right)$ is equivalent to ϕ. Likewise let $\mathfrak{A}^{-1}$ be a function from counter-free Büchi automata of the form $\mathcal{A} = \left(2^{\Phi}, S, S_0, \delta, F\right)$ to CTL* formulas, such that we have $\sigma \vDash \mathfrak{A}^{-1}\left(\mathcal{A}\right)$ iff $\mathcal{A}$ accepts $g_{\Phi}\left(\sigma\right)$.

4.2 Construction of $\tau_{\triangle}\left(\mathcal{A}\right)$ from $\mathcal{A}$.

In this section we define the function $\tau_{\triangle}$ from counter-free Büchi automata to automata as follows. The intention is that if $\mathcal{A}$ is equivalent to ϕ, then $\tau_{\triangle}\left(\mathcal{A}\right)$ will be equivalent to $\triangle\phi$. For any counter-free automata $\mathcal{A} = \left\langle 2^{\Phi}, S, S_0, \delta, F\right\rangle$, it is the case that $\tau_{\triangle}\left(\mathcal{A}\right) = \left\langle 2^{\Phi_{\triangle}}, S_{\triangle}, S_0, \delta_{\triangle}, F_{\triangle}\right\rangle$ where

1. $\Phi_\triangle = \Phi \cup \Psi$ where $\Psi = \{\psi_s : s \in S\}$ and ψ_s is the following state formula for each s in S:

$$E\left(\mathfrak{A}^{-1}\left(2^\Phi, S, \{s\}, \delta, F\right) \wedge NNG\neg\mathbf{v}\right)$$

ψ_s is roughly equivalent to saying "if we are in state s, we can deviate here".

2. We add a state s_F indicating that there existed an accepting deviation from this path and so we shall accept regardless of further input. This input relates to the original path rather than the deviation and is thus irrelevant. As such, $S_\triangle = S \cup s_F$ and $F_\triangle = F \cup s_F$.

3. $\delta_\triangle$ is the relation that includes δ but at each state also gives the option to branch into s_F when a deviation is possible and remain in that state regardless of the input along the current path. That is, $\delta_\triangle$ it the minimal relation satisfying:

   (a) For every tuple $\langle s, e, t\rangle$ in δ and set $\Theta \in 2^\Psi$ the tuple $\langle s, e \cup \Theta, t\rangle$ is in $\delta_\triangle$. This is to ensure that wherever $g_\Phi(\sigma)$ is a run of $\mathcal{A}$, it is also the case that $g_{\Phi_\triangle}(\sigma)$ is a run of $\tau_\triangle(\mathcal{A})$. The use of Θ is only required because $\delta_\triangle$ has to deal with the atoms in Ψ, which are not in $g_{\Phi_\triangle}(\sigma)$ but not in $g_\Phi(\sigma)$.

   (b) For each $s \in S$ and each $e_\triangle \in 2^{\Phi_\triangle}$ such that $\psi_s \in e_\triangle$ we have $\langle s, e_\triangle, s_F\rangle$ in $\delta_\triangle$.

   (c) For each $e_\triangle$ in $2^{\Phi_\triangle}$ we have $\langle s_F, e_\triangle, s_F\rangle$ in $\delta_\triangle$.

To understand (1) above, say we have read i symbols from the current path σ, and are in state s. Clearly if $\left\langle 2^\Phi, S, \{s\}, \delta, F\right\rangle$ accepts $\sigma_{\geq i}$ then then $\mathcal{A}$ accepts σ. As an i-deviation π would share the prefix $\sigma_{\leq i}$, if $\left\langle 2^\Phi, S, \{s\}, \delta, F\right\rangle$ accepts $\pi_{\geq i}$ then $\mathcal{A}$ accepts π. The $NNG\neg\mathbf{v}$ ensures that ψ_s is satisfied only on paths that deviate here, i.e. that are i-deviations.

The items (b) and (c) above allow us to ignore all further input on the current path if we can satisfy ϕ on a deviation.

Lemma 24. *The automaton $\tau_\triangle(\mathcal{A})$ is counter-free.*

Proof. Recall that a counter-free automaton is an automaton such that for all states $s \in S$ and words u in Σ^*, if $u^m \in L_{s,s}$ then $u \in L_{s,s}$.

If $s = s_F$ then every word u is in $L_{s,s}$. If $s \neq s_F$ then every path from s to s in $\tau_\triangle(\mathcal{A})$ is also a path from s to s in $\mathcal{A}$, and $\mathcal{A}$ is counter-free. □

4.3 Recursive Translation Function

We can now translate a RoCTL* formula ϕ into a CTL* formula $c(\phi)$ using the recursively defined function c:

$$\begin{aligned}
c(\phi \wedge \psi) &= c(\phi) \wedge c(\psi) \\
c(\neg\phi) &= \neg c(\phi) \\
c(A\phi) &= Ac(\phi) \\
c(O\phi) &= A(NG\neg\mathbf{v} \rightarrow c(\phi)) \\
c(N\phi) &= Nc(\phi) \\
c(\phi U \psi) &= c(\phi) U c(\psi) \\
c(\blacktriangle\phi) &= \neg f_\triangle(\neg\phi) \\
f_\triangle(\phi) &= \mathfrak{A}^{-1}(\tau_\triangle(\mathfrak{A}(\phi)))\ .
\end{aligned}$$

4.4 Proof of Correctness

Lemma 25. *For all structures M, fullpaths σ and CTL* formulas ϕ, it is the case that $M, \sigma \vDash \mathfrak{A}^{-1}(\tau_\triangle(\mathfrak{A}(\phi)))$ iff $M, \sigma \vDash \triangle\phi$.*

Proof. As above, let $\mathcal{A} = \left\langle 2^\Phi, S, S_0, \delta, F\right\rangle$ be $\mathfrak{A}(\phi)$; let $\tau_\triangle(\mathcal{A}) = \left\langle 2^{\Phi_\triangle}, S_\triangle, S_0, \delta_\triangle, F_\triangle\right\rangle$ be the automaton constructed from $\mathcal{A}$, and let $\phi_\triangle = \mathfrak{A}^{-1}(\tau_\triangle(\mathfrak{A}(\phi)))$.

Fix a structure M. We will write $M, \sigma \vDash \phi$ as $\sigma \vDash \phi$.

($\Longleftarrow$) Say that $\sigma \vDash \triangle\phi$.

Case 1: $\sigma \vDash \phi$; then $\mathcal{A}$ accepts $g_\Phi(\sigma)$. Thus $\tau_\triangle(\mathcal{A})$ accepts $g_{\Phi_\triangle}(\sigma)$ (see 3a above). Thus $\sigma \vDash \phi_\triangle$.

Case 2: $\sigma \nvDash \phi$; then there exists a path $\pi \vDash \phi$ and integer i such that $\sigma_{\leq i} = \pi_{\leq i}$ and $\pi_{\geq i+1}$ is failure free. Hence $\langle g_\Phi(\pi_0), g_\Phi(\pi_1), \ldots\rangle \in \mathcal{L}(\mathcal{A})$. Thus there exists a sequence of states $s_0, s_1, \ldots$ such that $s_0 \overset{g_\Phi(\pi_0)}{\rightarrow} s_1 \overset{g_\Phi(\pi_1)}{\rightarrow} \cdots$ is an accepting run for $\mathcal{A}$. It is easy to show that $s_i \overset{g_\Phi(\pi_i)}{\rightarrow} s_{i+1} \overset{g_\Phi(\pi_{i+1})}{\rightarrow} \cdots$ is an accepting run of $\left\langle 2^\Phi, S, \{s_i\}, \delta, F\right\rangle$. Thus $\pi_{i\geq i} \vDash \mathfrak{A}^{-1}\left(\left\langle 2^\Phi, S, \{s_i\}, \delta, F\right\rangle\right)$. As $\pi_{\geq i+1}$ is failure free $\pi_{\geq i} \vDash NNG\neg\mathbf{v}$, hence we have

$$E\left(\mathfrak{A}^{-1}\left(2^\Phi, S, \{s_i\}, \delta, F\right) \wedge NNG\neg\mathbf{v}\right) \quad = \quad \psi_{s_i} \in g_{\Phi_\triangle}(\pi_i)\ .$$

From 3a above, $s_0 \overset{g_{\Phi_\triangle}(\pi_0)}{\rightarrow} s_1 \overset{g_{\Phi_\triangle}(\pi_1)}{\rightarrow} \cdots \overset{g_{\Phi_\triangle}(\pi_{i-1})}{\rightarrow} s_i$

is a path through $\tau_\triangle(\mathcal{A})$. As $\psi_{s_i} \in g_{\Phi_\triangle}(\pi_i)$ it follows from 3b above that $\langle s_i, g_{\Phi_\triangle}(s_i), s_F\rangle \in \delta_\triangle$. Also $\langle s_F, e_\triangle, s_F\rangle \in \delta_\triangle$ for all $e_\triangle$ in 2^Φ. As $\sigma_{\leq i} = \pi_{\leq_i}$ it follows that $s_0 \overset{g_{\Phi_\triangle}(\sigma_0)}{\rightarrow} s_1 \overset{g_{\Phi_\triangle}(\sigma_1)}{\rightarrow} \cdots$ is an accepting run for $\tau_\triangle(\mathcal{A})$. Thus $\sigma \vDash \mathfrak{A}^{-1}(\tau_\triangle(\mathcal{A})) = \phi_\triangle$.

($\Longrightarrow$) Say that $M, \sigma \vDash \phi_\triangle$. Thus there is an accepting run $s_0 \overset{g_{\Phi_\triangle}(\sigma_0)}{\rightarrow} s_1 \overset{g_{\Phi_\triangle}(\sigma_1)}{\rightarrow} \cdots$ for $\tau_\triangle(\mathcal{A})$.

Case 1: $s_0 \overset{g_\Phi(\sigma_0)}{\rightarrow} s_1 \overset{g_\Phi(\sigma_1)}{\rightarrow} \cdots$ is an accepting run for $\mathcal{A}$. Then $\sigma \vDash \phi$ and so $\sigma \vDash \triangle\phi$.

Case 2: $s_0 \overset{g_\Phi(\sigma_0)}{\rightarrow} s_1 \overset{g_\Phi(\sigma_1)}{\rightarrow} \cdots$ is not an accepting run for $\mathcal{A}$. Thus the automaton must eventually enter state s_F, so the run has a prefix of the form $s_0 \overset{g_{\Phi_\triangle}(\sigma_0)}{\rightarrow} s_1 \overset{g_{\Phi_\triangle}(\sigma_1)}{\rightarrow} \cdots \overset{g_{\Phi_\triangle}(\sigma_{i-1})}{\rightarrow} s_i \overset{g_{\Phi_\triangle}(\sigma_i)}{\rightarrow} s_F$. We know from the definition of $\tau_\triangle$ above that $\psi_{s_i} \in g_{\Phi_\triangle}(\sigma_i)$. Thus $\sigma_{i\geq 1} \vDash E\left(\mathfrak{A}^{-1}(\Sigma, S, \{s_i\}, \delta, F) \wedge NNG\neg\mathbf{v}\right)$ and so there exists a path π such that $\pi_{\leq i} = \sigma_{\leq i}$, $\pi_{\geq i+1}$ is failure-free and $\pi_{\geq i} \vDash \mathfrak{A}^{-1}(\Sigma, S, \{s_i\}, \delta, F)$. It follows that $\langle \Sigma, S, \{s_i\}, \delta, F\rangle$ accepts $\pi_{\geq i}$ and so there exists an accepting run $s_i \overset{g(\pi_i)}{\rightarrow} s_{i+1} \overset{g(\pi_{i+1})}{\rightarrow} \cdots$. We see that the path $s_0 \overset{g(\sigma_0)}{\rightarrow} s_1 \overset{g(\sigma_1)}{\rightarrow} \cdots \overset{g(\sigma_{i-1})}{\rightarrow} s_i$ through $\tau_\triangle(\mathcal{A})$ is also a path through $\mathcal{A}$. Thus

$$s_0 \overset{g(\sigma_0)}{\rightarrow} s_1 \overset{g(\sigma_1)}{\rightarrow} \cdots s_i \overset{g(\pi_i)}{\rightarrow} s_{i+1} \overset{g(\pi_{i+1})}{\rightarrow} \cdots$$

is a run of $\mathcal{A}$. States in F occur in finitely often in $s_i, s_{i+1}, \ldots$ and hence this run is an accepting run of $\mathcal{A}$. Hence $\pi \vDash \phi$. As π is a deviation from σ it follows that $\sigma \vDash \triangle\phi$. □

Theorem 26. *There exists a satisfaction preserving translation function from RoCTL* to CTL*.*

Proof. Using Lemma 24 it is easy to show that the translation function c from Section 4.3 is well defined, and from Lemma 25 it is easy to show that c is satisfaction preserving. □

5. Succinctness

In the previous section we showed that a satisfaction preserving translation from RoCTL* to CTL* exists. In this section we will show that any satisfaction preserving translation is non-elementary in the length of the formulas.

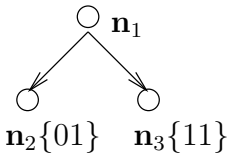
We will do this by taking a class of labeled trees which we will call (h, l)-utrees, where h represents the height h and l is the number of bits per label. We will show that the number $\#(h, l)$, of pairwise non-isomorphic (h, l)-utrees, is non-elementary in h. We will then present "suffix" and "prefix" encodings of utrees into RoCTL-structures, and will define a function u such that $u(T, T') = M$ where M is the structure that results when the prefix encoding of T is joined/followed by the suffix encoding of T'. For each positive h and l we define a RoCTL* formula $f(h, l)$ such that for any pair of utrees T and T' of height h it is the case that $u(T, T')$ satisfies $f(h, l)$ iff T, T' are isomorphic. For an automaton that accepts the tree-unwinding of $u(T, T')$ iff T and T' are isomorphic, once the automaton has read the prefix encoding, the state of the automaton must give us enough information to determine which of $\#(h, l)$ isomorphic equivalence classes T fell into. As $\#(h, l)$ is non-elementary in h, the number of states in the automata must also be non-elementary in h. Since there are elementary translations of CTL* into automata, we will conclude that there is no elementary translation of RoCTL* into CTL*.

Definition 27. We define *isomorphism* on finite labelled trees recursively. We say that $T = (A, R, g)$ and $T' = (A', R', g')$ are isomorphic if $g(\textbf{root}(T)) = g'(\textbf{root}(T'))$ and there exist orderings $\mathcal{C} = (C_1, \ldots, C_{|\mathcal{C}|})$ and $\mathcal{C}' = \left(C'_1, \ldots, C'_{|\mathcal{C}|}\right)$ of the direct subtrees of T and T' respectively such that C_i and C'_i are isomorphic for all $i \in [1, |\mathcal{C}|]$.

We define *utrees* below such that all (h, l)-utrees have the same number of direct subtrees, which are pairwise non-isomorphic. For any pair T, T' of (h, l)-utrees, this ensures that if there is a direct subtree of T that is not isomorphic to any subtree of T', there must also be a direct subtree of T' that is not isomorphic to any subtree of T. This makes it easier to test whether a pair of utrees are isomorphic.

Definition 28. We define the concept of a *utree* recursively. We fix an infinite enumerated set $\mathcal{V}_\omega = \{b_1, b_2, \ldots\}$. A tree $T = \langle A, R, g\rangle$ consisting of a single node $\mathbf{n}$ is a $(0, l)$-utree iff $g(\mathbf{n}) \subseteq \mathcal{V}_l$ where $\mathcal{V}_l = \{b_1, b_2, \ldots b_l\}$. We let $\#(h, l)$ be the number of pairwise non-isomorphic (h, l)-utrees; then a tree T is a $(h+1, l)$-utree iff $g(\textbf{root}(T)) = \emptyset$ and T has $\lfloor \#(h, l)/2 \rfloor$ direct subtrees, which are pairwise non-isomorphic (h, l)-utrees.

Example 29. Here is an example $(1,2)$-utree. We use "11" as shorthand for b_1, b_2 and "01" as shorthand for b_2.



Lemma 30. *The function $\#(h,l)$ is at least $(h+1)$-exponential in l.*

Proof. We see that the number of pairwise non-isomorphic $(0,l)$-utrees is 2^l. From the definition of utrees where $n = 2\lfloor \#(h,l)/2 \rfloor$,

$$\begin{aligned}\#(h+1,l) &\geq nC\left(\frac{n}{2}\right) = \frac{n!}{\frac{n}{2}!\frac{n}{2}!} \\ &= \frac{n.n-1\ldots\frac{n}{2}\ldots 2.1}{\left(\frac{n}{2}.\ldots.2.1\right)\left(\frac{n}{2}.\ldots.2.1\right)} \\ &= \frac{n(n-1)\ldots\left(\frac{n}{2}+1\right)}{\left(\frac{n}{2}.\ldots.2.1\right)} \geq 2^{\left(\frac{n}{2}\right)}.\end{aligned}$$

Thus when $\#(h,l)$ is j-exponential in l, it is the case that $\#(h+1,l)$ is $(j+1)$-exponential in l. As $\#(0,l)$ is one exponential in l it follows from induction that $\#(h,l)$ is at least $(h+1)$-exponential in l. □

Algorithm 1 can be used to describe the structure of a tree using a string of '{' and '}' characters; strictly speaking **for each** must iterate over the subtrees in some order, but the ordering chosen is unimportant and will not be defined here. E.g. "{}" represents a tree with a single node, and "{{}{}}" represents a tree where the root has two root nodes as successors.

Algorithm 1 PrintTree(T)

```
1: Print '{'
2: for each direct subtree D of T: PrintTree()
3: Print '}'
```

Algorithms 2 and 3 for outputting the prefix encoding **prefix** (T) of T are similar. The function **prefix** is from utrees to labelled trees where each node has degree of at most one. In addition to the atoms used to label the input tree, the prefix encoding also uses the following atoms as labels, where h is the height of the tree and $k \in [0,h]$.

$I_\{$ This atom indicates that we begin the description of a direct subtree of the tree we were describing. The current world also encodes the label of this subtree.

$I_\}$ This atom indicates that we are ending the description of some tree.

t_C This indicates that the description of the subtree C starts here. This is not used in function f below. It is only included to allow sections of the encoding to be easily and unambiguously referenced in the proof of correctness.

H_k The current input character describes the start of a tree of height k, we are at a node of height k. Thus $I_\{ \wedge H_3$ means we are beginning the definition of a tree of height 3 and $I_\} \wedge H_3$ means we are ending the definition of a tree of height 3.

The final world in the prefix encoding is w_Z; the prefix encoding is not a transition structure as w_Z has no successor.

Example 31. Below we present the prefix encoding of the utree T from Example 29.

$$\begin{array}{l} \bigcirc\ w_0\{I_\{, H_1, t_T\} \\ \Downarrow \\ \bigcirc\ w_1\{I_\{, H_0, 01, t_{(\mathbf{n}_2,\emptyset,\{\mathbf{n}_2\mapsto 01\})}\} \\ \Downarrow \\ \bigcirc\ w_2\{I_\}, H_0\} \\ \Downarrow \\ \bigcirc\ w_3\{I_\{, H_0, 11, t_{(\mathbf{n}_3,\emptyset,\{\mathbf{n}_3\mapsto 11\})}\} \\ \Downarrow \\ \bigcirc\ w_4\{I_\}, H_0\} \\ \Downarrow \\ \bigcirc\ w_5\{I_\}, H_1\} \\ \Downarrow \\ \bigcirc\ w_Z \end{array}$$

Algorithm 2 T2prefix(T)

```
1: (g, i):=T2g(T,∅,0)
2: A := domain (g) ∪ {w_Z}
3: → := {(w_{j-1}, w_j) : j ∈ [1, i)} ∪ (w_i, w_Z)
4: return(A, →, g)
```

We now define the suffix encoding **suffix** (T) of a tree $T = \left(A^T, R^T, g^T\right)$. In addition to the atoms used in the labelling of the input tree T, the suffix encoding uses: the

Algorithm 3 T2g(T,g,i)

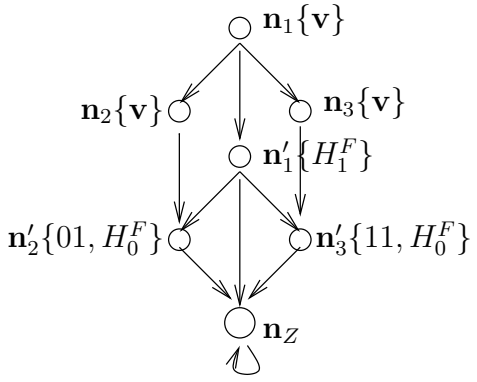
1: $\left(A^T, R^T, g^T\right)$:=$T$
2: $g[w_i]$:=$\{I_\{, H_{\textbf{height}(T)}, t_T\} \cup g^T(\textbf{root}(T))$; i++
3: **for each** direct subtree C of T: (g, i):=T2g(C,g,i)
4: $g[w_i]$:=$\{I_\}, H_{\textbf{height}(T)}\}$; i++
5: **return**(g, i)

violation atom $\mathbf{v}$ from RoCTL*; and H_k^F for k in $[0, h]$ which is used to indicate the height of the current node in the tree, much like H_k is used in the prefix encoding. Let $N = \left\{\mathbf{n}_1, \ldots, \mathbf{n}_{|N|}\right\}$ be the set of nodes in the tree T. Let N' be a numbered set such that $|N| = |N'|$; that is $N' = \left\{\mathbf{n}'_1, \ldots, \mathbf{n}'_{|N|}\right\}$. Then for all trees T, if $(A, R, g) = \textbf{suffix}(T)$ we have

1. $A = N \cup N' \cup \{\mathbf{n}_Z\}$
2. R is the minimal relation satisfying: $R \supseteq R^T$, and $\{(\mathbf{n}_i, \mathbf{n}'_i), (\mathbf{n}'_i, \mathbf{n}_Z), (\mathbf{n}_Z, \mathbf{n}_Z)\} \subseteq R$, for all $i \in [1, |N|]$.
3. the valuation g is the valuation satisfying $g(\mathbf{n}_i) = \{\mathbf{v}\}$; $g(\mathbf{n}_Z) = \emptyset$ and

$$g(\mathbf{n}'_i) = g^T(\mathbf{n}_i) \cup \left\{H^F_{\textbf{height}_{R^T}(\mathbf{n}_i)}\right\}.$$

Example 32. Below we present the suffix encoding of the utree from Example 29.



Definition 33. We let $u(T, T')$ be the model that results when we join the prefix encoding of T to the suffix encoding of T' by adding $(w_Z, \textbf{root}(T'))$ to R. Formally, where $\left(A^P, R^P, g^P\right)$ is the prefix encoding of T and $\left(A^S, R^S, g^S\right)$ is the suffix encoding of T', it is the case that $u(T, T') = (A, R, g)$ where $A = A^P \cup A^S$, $g(w) = g^S(w)$ if $w \in A^S$, $g(w) = g^P(w)$ if $w \in A^P$, $R = R^S \cup R^P \cup \{(w_Z, \textbf{root}(T'))\}$.

Definition 34. We say that a structure M *satisfies* a formula ϕ iff there exists a path σ such that $M, \sigma \vDash \phi$.

Definition 35. Let us define a function f as follows from pairs of natural numbers to RoCTL* formulas:

$$f(0, l) = \bigwedge_{i \in [1,l]} \left(b_i \rightarrow F\left(H_0^F \wedge b_i\right)\right) \wedge \bigwedge_{i \in [1,l]} \left(\neg b_i \rightarrow F\left(H_0^F \wedge \neg b_i\right)\right)$$

$$f(k, l) = \left(\left(I_\{ \wedge H_{k-1}\right) \rightarrow \triangle f(k-1, l)\right) U \left(I_\} \wedge H_k\right) \wedge F H_k^F \wedge \left(I_\{ \wedge H_k\right)$$

Recall that $F\phi$ is shorthand for $(\top U \phi)$, and as such $M, \sigma \vDash F\phi \iff \exists_i M, \sigma_{\geq i} \vDash \phi$.

The intuition behind f is that a path σ through $u(T, T') = \langle A, R, g\rangle$ can correspond to both a subtree of T and a subtree T'; if $t_C \in g(\sigma_0)$ then σ starts at the beginning of the prefix encoding of some subtree C of T, and if $\mathbf{n}'_{C'}$ is in σ then σ corresponds to some subtree C' of T'. The formula $f(0, l)$ is satisfied if the labels of C and C' match, so $f(0, l)$ is satisfied iff C and C' are isomorphic leafs. A deviation from the current path can only have one additional failure, and hence only one additional edge. So, where $\mathbf{n}'_{C'}$ is in σ, then for each subtree D' of T' satisfying $\textbf{height}(D') = \textbf{height}(C) - 1$ there exists a deviation from σ containing $\mathbf{n}'_{D'}$ iff D' is a direct subtree of C'. As such, $\triangle f(0, l)$ is satisfied exactly on those paths that correspond to subtrees C and D' such that C has a direct subtree isomorphic to D'. We use this intuition and recursion to prove the following lemma.

Lemma 36. *For any integers u and l, if T and T' are (u, l)-utrees then $u(T, T')$ satisfies $f(u, l)$ iff T and T' are isomorphic.*

Proof. For each subtree C of T, let w_C be the world that is the beginning of the suffix encoding of C, or more formally the world where t_C is true. For any path, σ we define $\sigma_{\geq C}$ such that $\sigma_{\geq C} = \sigma_{\geq i}$ where $\sigma_i = w_C$.

($\Longrightarrow$) Say that $u(T, T'), \sigma^T \vDash f(u, l)$ for some σ^T. We see that $\sigma_0^T = w_0$ as $f(u, l) \vDash I_\{ \wedge H_u$. We define σ^C

recursively for each subtree C of T. Say we have defined the path σ^C for some subtree C such that $u\left(T,T'\right),\sigma^C \vDash f\left(k,l\right)$ where k is the height of C. Then for each direct subtree D of C, we see that $\sigma^C_{\geq D} \vDash \triangle f(k-1,l)$ and thus there must exist a deviation from $\sigma^C_{\geq D}$ satisfying $f(k-1,l)$, we call this deviation σ^D.

We see that for each C there is a unique C' such that $\mathbf{n}'_{C'}$ is in the path σ^C. In the following paragraph we will show that for each subtree C and direct subtree D of C, we can produce σ^D from $\sigma^C_{\geq D}$ by replacing $\mathbf{n}'_{C'}$ with $\mathbf{n}_{D'}\mathbf{n}'_{D'}$, and hence that D' is a direct subtree of C'.

Consider where σ^D deviates from $\sigma^C_{\geq D}$. Say $\mathbf{n}_y$ is the first world in σ^D not in σ^C and that $\mathbf{n}_x$ is the last world in both σ^C and σ^D. From the definition of deviations we see that $\sigma^D_{\geq \mathbf{n}_y}$ is failure-free and so the next world on σ^D must be $\mathbf{n}'_B$. Since $\sigma^D \vDash FH^F_k$ where k is the height of D it follows that $H^F_k \in g\left(\mathbf{n}'_B\right)$; from the structure of the suffix encoding it is clear that B is a direct subtree of A, and **height** $(A) = k+1$ and thus H^F_{k+1} in $g\left(\mathbf{n}'_A\right)$. As each parent has a height greater than that of its direct subtrees, it follows that $\mathbf{n}_{C'}$ is the only world in σ^C such that $H^F_{k+1} \in \mathbf{n}'_{C'}$, and hence it follows that $\mathbf{n}_x = \mathbf{n}_{C'}$.

Consider D of height 0. The path σ^D is of the form

$$\langle w_D, \ldots, w_Z, \mathbf{n}_T, \ldots, \mathbf{n}_{C'}, \mathbf{n}_{D'}, \mathbf{n}'_{D'}, \mathbf{n}_Z, \mathbf{n}_Z, \ldots\rangle$$

It is easy to show that D and D' are isomorphic. For each C, we choose C' such that $\mathbf{n}'_{C'}$ is in the full path σ^C. Say that for every D of height k it is the case that D' and D are isomorphic. Consider C of height $k+1$. We have shown that for each direct subtree D of C, it is the case that D' is a direct subtree of C'. As C must have the same height as C' (otherwise the requirement that $\sigma \vDash FH^F_{k+1}$ would not be satisfied), C' and C have the same number of direct subtrees, each of height k. We have show previously that for each direct subtree D of C, it is also the case that D' is a direct subtree of C'. By assumption, each pair D, D' are isomorphic, and so C, C' are isomorphic. By induction T and T' are isomorphic.

($\Longleftarrow$) Say that T' and T are isomorphic. Clearly suffix encodings of T' and T will also be isomorphic, and so $u\left(T,T'\right)$ satisfies $f\left(u,l\right)$ iff $u\left(T,T\right)$ does. Thus we can assume without loss of generality that $T = T' = \left(A^T, R^T, g^T\right)$.

Likewise let $\mathbf{n}_C$ be the node that is the root of the subtree C. We define σ^C recursively as follows: let σ^T be the fullpath starting at w_0 that passes through $\mathbf{n}'_0$; that is, $\sigma^T = \langle w_0, \ldots, w_Z, \mathbf{n}_T, \mathbf{n}'_T, \mathbf{n}_Z, \mathbf{n}_Z, \ldots\rangle$. Say that D is the direct subtree of C, then where

$$\sigma^C = \langle w_C, \ldots, w_D, \ldots, w_Z, \mathbf{n}_T, \ldots, \mathbf{n}_C, \mathbf{n}'_C, \mathbf{n}_Z, \mathbf{n}_Z, \ldots\rangle$$

we let

$$\sigma^D = \langle w_D, \ldots, w_Z, \mathbf{n}_T, \ldots, \mathbf{n}_C, \mathbf{n}_D, \mathbf{n}'_D, \mathbf{n}_Z, \mathbf{n}_Z, \ldots\rangle \ .$$

In other words, we produce $\sigma^C_{\geq D}$ from σ^C by pruning everything prior to w_D, and σ^D from $\sigma^C_{\geq D}$ and replacing $\mathbf{n}'_C$ with $\mathbf{n}_D, \mathbf{n}'_D$. This remains a full path, since D is a direct subtree of C, and so $\mathbf{n}_D$ is a child of $\mathbf{n}_C$. Note also that σ^D is a deviation from $\sigma^C_{\geq D}$.

If **height** $(C) = 0$ it is easy to verify that $\sigma^C \vDash f\left(0,l\right)$, as $g\left(w_C\right) \cup \left\{H^F_0\right\} = g\left(\mathbf{n}_C\right) \cup \left\{H_0, t_C, I_{\{}\right\}$. For C of height k, it is likewise easy to see that $\sigma^C \vDash FH^F_k$. Assume that $\sigma^C \vDash f\left(k-1,l\right)$ for all C of height $k-1$ [expand]. Now consider C of height k. It is easy to show that

$$\sigma^C \vDash \left(\left(I_{\{} \wedge H_{k-1}\right) \rightarrow \bigvee_{D \text{ is child of C}} t_D\right) U \left(I_{\}} \wedge H_k\right) \ .$$

By assumption $\sigma^D \vDash f(k-1,l)$, and σ^D is a deviation from $\sigma^C_{\geq D}$, so $\sigma^C_{\geq D} \vDash \triangle f(k-1,l)$. Thus

$$\sigma^C \vDash \left(\left(I_{\{} \wedge H_{k-1}\right) \rightarrow \triangle f(k-1)\right) U \left(I_{\}} \wedge H_k\right) \ .$$

Thus $\sigma^C \vDash f\left(k,l\right)$. By induction $u\left(T,T'\right),\sigma^T \vDash f\left(u,l\right)$.

□

Example 37. In Lemma 36 above, we proved that $u\left(T,T'\right),\sigma^T \vDash f\left(u,l\right)$ for some σ^T iff T and T' are isomorphic. Using T as the tree in Example 29, let

$$\sigma^0 = \langle w_0, \ldots, w_Z, \mathbf{n}_1, \mathbf{n}'_1, \mathbf{n}_Z, \ldots\rangle$$
$$\sigma^1 = \langle w_1, \ldots, w_Z, \mathbf{n}_1, \mathbf{n}_2, \mathbf{n}'_2, \mathbf{n}_Z, \ldots\rangle$$
$$\sigma^2 = \langle w_3, \ldots, w_Z, \mathbf{n}_1, \mathbf{n}_3, \mathbf{n}'_3, \mathbf{n}_Z, \ldots\rangle$$

be paths through $u\left(T,T\right)$. We see that σ^1 and σ^2 sat-

isfy $f(0,2)$. As σ^1 and σ^2 are deviations from $\sigma^0_{\geq 1}$ and $\sigma^0_{\geq 3}$ respectively, it is the case that $\sigma^0_{\geq 1}$ and $\sigma^0_{\geq 3}$ satisfy $\triangle f(0,2)$. Thus wherever $I_{\{} \wedge H_0$ is true, it is also the case that $\triangle f(0,2)$ is true; hence $u(T,T), \sigma^0 \vDash f(1,2)$.

Definition 38. We say an automaton $\mathcal{A}$ accepts a structure M iff the tree unwinding of M is a member of $\mathcal{L}(\mathcal{A})$.

Lemma 39. *For any arbitrary $h, l \in \mathbb{N}$, let $\mathcal{A} = (\Sigma, S, S_0, \delta, F)$ be an SAA such that for any pair T, T' of (h,l)-utrees $\mathcal{A}$ accepts $u(T,T')$ iff T and T' are isomorphic; then $2^{|S|} \geq \#(h,l)$.*

Proof. Let $\{T_1, T_2, \ldots, T_{\#(h,l)}\}$ be a set of pairwise non-isomorphic (h,l)-utrees. For each i, let $R_i = \langle A_{R_i}, g_{R_i}, R_{R_i} \rangle$ be an accepting run of $\mathcal{A}$ on $u(T_i, T_i)$; let Q_i be the set of all states that the automata is in after reading the prefix encoding of T_i; formally let $Q_i \subseteq S$ be the set of states such that for all $q \in S$ we have $q \in Q_i$ iff there exists $w_R \in A_{R_i}$ such that $(\textbf{root}(T_i), q) \in g_{R_i}(w_R)$. Recall that $\textbf{root}(T_i)$ is the beginning of the suffix encoding of $u(T_i, T_i)$.

Say that $Q_i = Q_j$ for some $i \neq j$. Let $\mathcal{A}_q$ be shorthand for $(\Sigma, S, \{q\}, \delta, F)$. In the next paragraph we will define a run R^i_j with the prefix from the run R_j and the suffix from R_i.

Since all infinite paths of the run R_i are accepting, we see that for each $q \in Q_i$, the relevant subtree $R^{\text{suffix}}_{i,q}$ of R_i is an accepting run for $\mathcal{A}_q$ on the suffix encoding of T_i. Let R^i_j be the tree that results when we replace the subtree beginning at w_R with $R^{\text{suffix}}_{i,q}$, for each $q \in Q_i = Q_j$ and $w_R \in A_{R_i}$ satisfying $g_{R_j}(w_R) = (\textbf{root}(T_j), q)$. It is easy to show that R^i_j is an accepting run of $\mathcal{A}$ on $u(T_j, T_i)$. However we have assumed that T_i is not isomorphic to T_j, and so $\mathcal{A}$ does not accept $u(T_j, T_i)$. By contradiction $Q_i \neq Q_j$ for any $i, j \in [1, \#(h,l)]$ such that $i \neq j$. As each $Q_i \in 2^S$, we can conclude from the pigeon hole principle that $2^{|S|} \geq \#(h,l)$. □

Theorem 40. *Given a CTL* formula ψ we can construct an SAA $\mathcal{A}_\psi$ with a number of states that is singly exponential in the length of ψ.*

Proof. Dam provides a translation of CTL* formulas into equivlent μ-calculus. The nodes are sets of formulas, so this is a singly exponential translation.

There are a number of translations of μ-calculus into alternating automata, Wilke gives a simple translation that does not assume that the tree has any particular structure [24]. The states in the resulting automata are subformulas of the μ-calculus formula. Hence the translation into alternating automata is linear. □

The translation via μ-calculus above is sufficient for this paper. There are translations that result in more optimised model checking and decision procedure results [15].

Corollary 41. *For all fixed $h \geq 1$, there is no function e which is less than $(h-1)$-exponential, such that the length $|\phi_l|$ of the shortest CTL* formula $\phi_l \equiv f(h,l)$ satisfies $|\phi_l| < e(l)$ for all l.*

Proof. Say e exists. Since $\phi_l \equiv f(h,l)$ then there exists a fullpath σ^T starting at w_0 through $u(T,T')$ such that $u(T,T'), \sigma^T \vDash \phi_l$ iff T and T' are isomorphic. As e is less than $(h-1)$-exponential, from Theorem 40 the size of the SAA is less than h-exponential in l.

From Lemma 39, we have $2^n \geq \#(h,l)$ where n is the size of the automata, and from Lemma 30 we know that $\#(h,l)$ is $(h+1)$-exponential in l. Hence 2^n is at least $(h+1)$-exponential in l, and so n is at least h-exponential in l. By contradiction no such e exists. □

Corollary 42. *For all fixed $h \geq 2$, there is no function e which is less than $(h-2)$-exponential such that for all RoCTL* formulas ϕ with at most h nested $\triangle$ (or $\blacktriangle$), the length $|\psi|$ of the shortest CTL* formula ψ equivalent to ϕ is no more than $e(|\phi|)$.*

Proof. This follows from the above corollary, and the fact that $f(h,l)$ has at most h nested $\triangle$ and $|f(h,l)| \in \mathcal{O}(h+l)$. □

Theorem 43. *There is no satisfaction preserving translation from RoCTL* to CTL* that is elementary in the length of the formula.*

Obvious from the above Corollary; if there were an i-exponential translation of RoCTL* into CTL* for any $i \in \mathbb{N}$ there would be an i-exponential translation of RoCTL* formulas with $i+3$ nested $\triangle$ operators.

We see that the only non-classical operators in $f(h,l)$ are positively occurring Δ, U and F. Since $F\psi$ is short

hand for $\top U \psi$ we see that alternations between positively occurring U and $\triangle$ are sufficient to produce non-elementary blowup. By slightly modifying f, we can similarly demonstrate that alternation between positively occurring $\blacktriangle$ and U are also sufficient to produce non-elementary blowup. For example the following f' contains only operators equivalent to negatively occurring U, where W is the weak until operator and $H^F \approx \bigvee_i H_i^F$:

$$f'(0,l) = \bigwedge_{i \in [1,l]} \left(b_i \rightarrow G\left(H^F \rightarrow \left(H_0^F \wedge b_i\right)\right)\right) \wedge \bigwedge_{i \in [1,l]} \left(\neg b_i \rightarrow G\left(H^F \rightarrow \left(H_0^F \wedge \neg b_i\right)\right)\right)$$

$$f'(k,l) = \left(\left(I_{\{} \wedge H_{k-1}\right) \rightarrow \triangle f'(k-1,l)\right) W \left(I_{\}} \wedge H_k\right) \wedge F H_k^F \wedge \left(I_{\{} \wedge H_k\right)$$

Since there is no elementary translation of f and f' into CTL*, there is also no elementary translation of $\neg f$ and $\neg f'$ into CTL*.

6. Conclusion

We have shown that all RoCTL* formulas can be expressed as an equivalent CTL* formula. This translation can also be used to translate RoBCTL* [17] formulas into BCTL* formulas. Once translated into CTL* formula we can use any of the standard methods for model checking, so this result provides us with a model checking procedure for RoCTL*. As with CTL*, the model checking problem for RoCTL* is linear with respect to the size of the model [8]. Classes of RoCTL* formulas with bounded $\blacktriangle$-complexity have linear translations into CTL*. Thus as with CTL* the model checking problem is also singly exponential [8] with respect to the length of these formulas , and satisfiability is doubly exponential. Multiple nestings of $\blacktriangle$ (or $\triangle$) without any form of alternation can also be translated to CTL* without increasing the complexity of the translation over a single $\blacktriangle$ operator.

We have not shown the exact complexity of the translation. However we will attempt to show that there is roughly a single exponential blowup per alternation between Δ (or $\blacktriangle$) and U; never-the-less we expect model checking to be practical for some useful subclasses of RoCTL* formulas. While in other logics non-elementary blowup is frequently the result of unbounded alternations between positive and negative occurrences of the same operator, we do not need to alternate between $\triangle$ and $\blacktriangle$ to demonstrate non-elementary blow up. Indeed, the only non-classical operators in the function f were positively occurring U and $\triangle$. We may modify f slightly so that it only contains positively occurring U and $\blacktriangle$.

RoCTL* is known to be decidable, but without a known elementary upper bound. Our succinctness result shows that a full translation into CTL* or Tree Automata cannot result in elementary decision procedures. The question still remains as to whether some other elementary decision procedure can be found for RoCTL*. The discovery of such a procedure would be interesting, as this would be the first modal logic which was elementary to decide but had only non-elementary translations into tree automata.

References

[1] H. Aldewereld, D. Grossi, J. Vazquez-Salceda, and F. Dignum. Designing normative behaviour by the use of landmarks. In *Agents, Norms and Institutions for Regulated Multiag. Syst.*, Utrecht, The Netherlands, Jul 2005.

[2] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *TCS*, 303(1):7–34, 2003.

[3] N. Belnap. Backwards and forwards in the modal logic of agency. *Philos. Phenomen. Res.*, 51(4):777–807, Dec 1991.

[4] J. Broersen, F. Dignum, V. Dignum, and J.-J. C. Meyer. *Designing a Deontic Logic of Deadlines*, volume 3065/2004 of *LNCS*, pages 43–56. Springer, 2004.

[5] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2):244–263, 1986.

[6] M. de Weerdt, A. Bos, H. Tonino, and C. Witteveen. A resource logic for multi-agent plan merging. *Annals of Math. and AI*, 37(1-2):93–130, January 2003.

[7] V. Diekert and P. Gastin. First-order definable languages. In J. Flum, E. Grädel, and T. Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.

[8] J. Edmund M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[9] E. A. Emerson and C.-L. Lei. Modalities for model checking (extended abstract): branching time strikes back. In *POPL '85: Proc. 12th ACM SIGACT-SIGPLAN symp. on Principles of programming languages*, pages 84–96, New York, NY, USA, 1985. ACM.

[10] J. W. Forrester. Gentle murder, or the adverbial samaritan. *J. Philos.*, 81(4):193–7, April 1984.

[11] T. French, J. C. M^{c}Cabe-Dansted, and M. Reynolds. *A Temporal Logic of Robustness*, volume 4720 of *LNCS*, pages 193–205. 2007. http://dx.doi.org/10.1007/978-3-540-74621-8_13.

[12] E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. 2002. LNCS, Vol. 2500 http://www.springer.com/computer/book/978-3-540-00388-5.

[13] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[14] T. Jéron, H. Marchand, S. Pinchinat, and M.-O. Cordier. Supervision patterns in discrete event systems diagnosis. In *8th Internat. Workshop on Discrete Event Syst.*, pages 262–268, July 2006.

[15] O. Kupferman and M. Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In *Proc CAV'00, LNCS*, volume 1855, pages 36–52. Springer, 2000.

[16] W. Long, Y. Sato, and M. Horigome. Quantification of sequential failure logic for fault tree analysis. *Reliab. Eng. Syst. Safe.*, 67:269–274, 2000.

[17] J. C. McCabe-Dansted. A tableau for RoBCTL*. In S. Hölldobler, C. Lutz, and H. Wansing, editors, *JELIA*, volume 5293 of *LNCS*, pages 298–310. Springer, 2008.

[18] L. T. McCarty. Defeasible deontic reasoning. *Fundam. Inform.*, 21(1/2):125–148, 1994.

[19] J. C. M^{c}Cabe-Dansted, T. French, and M. Reynolds. A temporal logic of robustness, RoCTL*. Technical report, UWA, 2007. http://www.csse.uwa.edu.au/~john/papers/RoCTL07.pdf.

[20] A. Pancones. The coordinated attack and the jealous amazons. http://www.dsi.uniroma1.it/~asd3/dispense/attack+amazons.pdf.

[21] A. Rodrigo and A. Eduardo. Normative pragmatics for agent communication languages. In *Perspect. Concept. Model. (LNCS)*, volume 3770, pages 172–181. Springer, 2005.

[22] F. van der Grijn. (im)possibility of a coordinated attack. Technical report, University of Amsterdam, June 2004. http://www.illc.uva.nl/Publications/ResearchReports/X-2004-05.text.pdf.

[23] L. W. N. van der Torre and Y. Tan. The temporal analysis of Chisholm's paradox. In T. Senator and B. Buchanan, editors, *Proc. 14th Nation. Conf. on AI and 9th Innov. Applic. of AI Conf.*, pages 650–655. AAAI Press, 1998.

[24] T. Wilke and C. albrechts-universitt Zu Kiel. Alternating tree automata, parity games, and modal μ-calculus, 2000.