On the Expressive Power of the Normal Form for Branching-Time Temporal logics

Alexander Bolotov

University of Westminster
London, UK
School of Computer Science and Engineering
A.Bolotov@wmin.ac.uk

With the emerging applications that involve complex distributed systems branching-time specifications are specifically important as they reflect dynamic and non-deterministic nature of such applications. We describe the expressive power of a simple yet powerful branching-time specification framework – branching-time normal form, which has been developed as part of clausal resolution for branching-time temporal logics. We show the encoding of Büchi Tree Automata in the language of the normal form, thus representing, syntactically, tree automata in a high-level way. Thus we can treat BNF as a normal form for the latter. These results enable us (1) to translate given problem specifications into the normal form and apply as a verification method a deductive reasoning technique – the clausal temporal resolution; (2) to apply one of the core components of the resolution method - the loop searching to extract, syntactically, hidden invariants in a wide range of complex temporal specifications.

1 Introduction

Automata theoretic methods are fundamental in study of branching-time logics, they are widely used in the investigations into the expressiveness of branching-time formalisms and their decidability, and are in the centre of the model-checking techniques [21], being widely used in the framework of formal verification. With the emerging applications that involve complex distributed systems deployed in heterogeneous environment, for example, robotics systems [20], branching-time specifications are becoming even more important. These specifications reflect the dynamic and non-deterministic nature of such applications and one can envisage the obvious need for efficient techniques to reason about them [10]. Tree structures are the underlying models for these specifications and tree automata are well established techniques to reason about tree structures, again, widely used in modern model-checking. On the other hand, it is useful to have direct methods of deductive reasoning applied to temporal specifications that enable carrying proof. Clausal temporal resolution is one of such techniques [10]. The method is based on the concept of the separated normal form initially developed for resolution based deductive verification in the linear-time framework (see full account of the method in [18] and an overview of various developments within this framework in [10]) and later extended to Computation Tree Logic CTL [2], [6], ECTL [4] and ECTL+ [3]. These branching-time logics differ in their expressiveness, with CTL allowing only a single temporal operator preceded by a path quantifier, ECTL extending CTL by allowing combinations of temporal operators that express fairness constraints and ECTL+ allowing Boolean combinations of temporal operators and ECTL fairness constraints (but not permitting their nesting). Yet, the same formalism serves as the normal form for all these logics. For simplicity, here we call this formalism BNF. The refinement and implementation of the clausal resolution in branching-time setting is given in [22, 23].

In [7] it was shown that the normal form for linear time is as expressive as Büchi automata on infinite words [9] while [11] defined the translation from an alternating automaton to this normal form and vice versa. In this paper we consider the analogous problem, in the branching-time setup, namely we show how BNF can represent, *syntactically*, Büchi tree automata in a high-level way. Note that in translating a branching-time problem specification into BNF, we actually derive clauses within a fragment of Quantified Computation Tree Logic (EQCTL) [19]. In particular, formulae within BNF are existentially quantified, moreover, this quantification is external to CTL formulae themselves. Thus, clauses of the BNF satisfy the criteria of the prenex normal form ([19]), and, translating the given branching-time specification into BNF we stay within this fragment. In order to utilise the normal form as part of proof in the branching-time framework, we effectively skolemize the normal form producing its clauses – temporal formulae – without any quantification.

Having established this relationship between BNF and Büchi tree automata, we justify, theoretically, that BNF is applicable to a wide class of branching-time specifications, and a resolution based verification technique can be used as reasoning tool for the obtained specification. Another important contribution of this paper is that BNF specifications obtained from the initial automaton give a good insight into the temporal context, as these specifications are defined with the use of the standard future time temporal operators: $\Box$ (always), $\bigcirc$ (next time), $\Diamond$ (eventually) and, additionally, path quantifiers A (on all future paths) and E (on some future path). Finally, we note that the result of this paper enables one of the core components of the resolution method - the loop searching ([5, 10]) - to be used to extract (again syntactically) hidden invariants in a wide range of complex temporal specifications.

The rest of the paper is organised as follows. In §2 we define the main technical terms of tree structures relevant to the paper. In §3 we overview BNF outlining its syntax in §3.1 and its semantics, in §3.2, together with an example, §3.3. In §4 we define Büchi tree automata. In §5, we show how to translate these automata into BNF and establish the correctness while in §6 the reverse translation from BNF into Büchi tree automata and its correctness are given. In §7 we give an example of the syntactical representation of a small Büchi tree automaton in BNF. Finally, in §8, we provide concluding remarks and discuss future work.

2   Tree Structure Notation

In this section we introduce main concepts of tree structures that are needed for the definition of Büchi Automata and BNF.

Definition 1 (Paths and Fullpaths of a Tree) *Given a* **tree** $T = (N, E)$ *such that N is a set of states (nodes) and $E \subset N \times N$ is a set of edges, with the root $x_0 \in N$, a path π_{x_i} of a tree T is a (possibly infinite) sequence of nodes $x_i, x_{i+1}, x_{i+2} \ldots$, where for each j ($i \le j$), $(x_j, x_{j+1}) \in E$. A path π_{x_0} of a tree with the root x_0 is called a* **fullpath**. *We will denote the set of all paths of some tree T by Π_T and the set of all fullpaths by X_T.* ∎

Given a tree $T = (N, E)$ the edge relation E is called *total* (or connected) if each node $x_i \in N$ belongs to some fullpath. We will concentrate on such trees where each node is connected with the root. Now the connectivity property of a tree (N, E) can be viewed as the following requirement: for any state $x_i \in N$ there must be a fullpath π_{x_0} such that $x_i \in \pi_{x_0}$, i.e. a path exists which connects this node x_i with the root x_0 of a tree.

Definition 2 (Prefix and Suffix of a path)   *Given a path, χ_{x_i} of a tree T and a node $x_j \in \chi_{x_i}$, where $i \le j$, we call a finite sub-sequence of nodes $[x_i, x_j] = x_i, x_{i+1}, \ldots x_j$ a* **prefix** *of a path χ_{x_i} abbreviating*

*it with $Pref(\chi_{x_i},[x_i,x_j])$ (or simply as $[x_i,x_j]$ when it is clear which path this prefix belongs to) and an infinite sub-sequence of nodes $x_j,x_{j+1},x_{j+2},\ldots$ a **suffix** of a path χ_{x_i} abbreviating it with $Suf(\chi_{x_i},x_j)$.* ∎

We will utilise the concepts of prefix and suffix in defining the closure properties below.

Closure properties of CTL models. When trees are considered as models for distributed systems, paths through a tree are viewed as computations. The natural requirements for such models would be suffix and fusion closures. Following [12], the former means that every suffix of a path is itself a path. The latter requires that a system, following the prefix of a computation γ, at any point $s_j \in \gamma$, is able to follow any computation π_{s_j} originating from s_j.

Finally, we require that "if a system can follow a path arbitrarily long, then it can be followed forever" [12]. This corresponds to limit closure property, meaning that for any fullpath γ_{s_0} and any paths $\pi_{s_j}, \phi_{s_k},\ldots$ such that γ_{s_0} has the prefix $[s_0,s_j]$, π_{s_j} has the prefix $[s_j,s_k]$, ϕ_{s_k} has the prefix $[s_k,s_l]$, etc, and $0 < j < k < l$, the following holds: there exists an infinite path α_{s_0} that is a limit of the prefixes $[s_0,s_j],[s_j,s_k],[s_k,s_l],\ldots$. Closure properties, especially, limit closure, are reflected in the formulation of the BNF, namely, in the introduction of labels for the clauses of the BNF.

Definition 3 (Labelled tree) *Given a tree $T = (N,E)$ and a finite alphabet, Σ, a Σ-labelled tree is a structure (T,L) where L is a mapping $N \longrightarrow \Sigma$, which assigns to each node, element of N, some label, element of Σ.* ∎

Definition 4 (Branching degree of a node, Branching factor of a tree structure) *The number, d, of immediate successors of a node x in a tree structure is called the* branching degree *of x. Thus, $x \cdot k$ $(1 \leq k \leq d)$ abbreviates the $k-th$ successor of x.*

Given a set $D = \{d_1,d_2,\ldots\}$, of the branching degrees of the nodes of a tree structure, the maximal d_i $(1 \leq i)$ is called the branching factor *of this tree structure. A tree structure with its branching factor d is called a d-ary tree structure.* ∎

We assume that underlying tree models are of at most countable branching factor. However, following ([12], page 1011) trees with arbitrary, even uncountable, branching, 'as far as our branching temporal logic are concerned, are indistinguishable from trees with finite, even bounded, branching'. This makes tree automata applicable in the branching-time framework. We will also use this result to justify the labelling of BNF clauses.

Definition 5 (ω-tree) *An ω-tree (N,E^ω) is a tree, which satisfies the following conditions.*

1. *A tree is of at most countable branching.*

2. *The relation E is* serial, *i.e. every state s_i must have at least one successor state.*

3. *E induces the natural ordering $\leq$: if $(s_i,s_j) \in E^\omega$ then $i \leq j$, where $\leq$ orders the set of natural numbers $\omega = \{0,1,2,\ldots\}$.* ∎

Now, following [15], given that a CTL model structure $\mathscr{M}$ (see §3.2 for the definition of a model structure) has its branching factor at most d, there exists a d-ary tree canonical model $\mathscr{M}'$ such that for any formula A, $\mathscr{M}$ satisfies A if, and only if, $\mathscr{M}'$ satisfies A. Informally, a canonical model is an unwinding of an arbitrary model $\mathscr{M}$ into an infinite tree T [15]. One of the essential properties of canonical models, which we will utilise here, is that the number of successors for every state is canonicalised by d.

3   Normal form used for the clausal resolution for CTL-type logics

Normal form BNF which we are considering, is a formalism that has been developed as part of the clausal resolution method developed initially for linear-time temporal logic [17, 18] and then defined for branching-time temporal logics, CTL [6] and its extensions, ECTL [4] and ECTL$^+$ [3][1]

As one would expect from clausal resolution, formulae of a given logic are first translated into normal form, which is a collection of clauses, to which a resolution method is applied. The idea behind the resolution procedure in temporal context is to extract for some given formula *A* a set of clauses that capture three types of 'knowledge' about *A* - what is happening at the beginning of the computation, what are the 'steps' of the computation, and what are the eventualities to be fulfilled during the computation.

For a formula *A* of a CTL-type branching-time logic, we will abbreviate its clausal normal form as BNF(*A*). Note that the standard formulation of CTL-type logics is based upon classical connectives, future time temporal operators $\bigcirc$ and $\mathscr{U}$ (until) (which is sufficient to introduce $\square$ and $\diamondsuit$) and path quantifiers A and E. The BNF, however, is formulated using $\bigcirc$, $\square$ and $\diamondsuit$, and specific to the BNF language, the operator **start** (which is only true at the beginning of the computation) but not utilising the $\mathscr{U}$ operator as it is removed during the translation to the normal form based on its fixpoint definition [6].

3.1   Language of BNF

In the language for BNF we utilise

- classical connectives: implication ($\Rightarrow$), negation ($\neg$), disjunction ($\vee$), and conjunction ($\wedge$);

- classically defined constants true (**T**) and false (**F**);

- temporal operators 'at the initial moment of time' (**start**), eventually ($\diamondsuit$), always ($\square$), next time ($\bigcirc$), and,

- path quantifiers: on all future paths (A) and on some future path (E).

In the rest of the paper we will use the following notation: **T** abbreviates any unary BNF temporal operator and **P** either of path quantifiers; any formula of the type **PT** is called *a basic* BNF *modality*, and a *literal* is a proposition or its negation.

Indices. The language for indices is based on the set of terms $\{IND\} = \{\mathsf{f, g, h}, \ldots\}$, where $\mathsf{f, g, h} \ldots$ denote constants. Note that indices play essential role in the formulation of BNF as they help identifying a specific path context for given formulae. We use indices to label all formulae of BNF that contain the basic modality $\mathsf{E}\bigcirc$ or $\mathsf{E}\diamondsuit$. Specifically, the modality $\mathsf{E}\bigcirc$ is associated with BNF step clauses (see below) and, thus, if $\mathsf{E}\bigcirc A_\mathsf{g}$ is true at some current state s_i then *A* holds at the successor state of s_i along the path associated with the 'direction' g - speaking informally, we only take 'one step' in direction *g* from s_i. The modality $\mathsf{E}\diamondsuit$ is associated with evaluating eventualities over a longer period of time, and thus if $\mathsf{E}\diamondsuit p_\mathsf{g}$ is true at some state s_i then *p* is true at some state s_k along the path which goes from s_i by making at s_i, and every subsequent successor state, a 'mini-step' along the 'direction' g. This corresponds to the limit closure of the concatenation of these 'mini-steps' in directions g and hence the existence of such a path is always guaranteed.

---

[1]Note the formulation of the clausal resolution technique for the logic CTL*, often referred as 'full branching-time logic', is still an open problem. This development is complicated by the problem of capturing the temporal path context in the BNF due to the unrestricted syntax of CTL* where any concatenation of temporal operators is allowed.

Definition 6 (Branching Normal Form) *Given Prop, a set of atomic propositions, and IND, a count-able set of indices,* BNF *has the structure*

$$\mathsf{A}\square\left[\bigwedge_i C_i\right]$$

where each of the clauses C_i *is defined as below and each* α_i, β_j *or* γ *is a literal,* T *or* F *and* ind $\in$ *IND is some index.*

$$
\begin{array}{rcll}
\mathbf{start} & \Rightarrow & \bigvee_{j=1}^{k} \beta_j & \text{an Initial Clause} \\[2ex]
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathsf{A}\bigcirc\left[\bigvee_{j=1}^{k} \beta_j\right] & \text{an A step clause} \\[2ex]
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathsf{E}\bigcirc\left[\bigvee_{j=1}^{k} \beta_j\right]_{\text{ind}} & \text{a E step clause} \\[2ex]
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathsf{A}\diamondsuit\gamma & \text{an A sometime clause} \\[2ex]
\bigwedge_{i=1}^{l} \alpha_i & \Rightarrow & \mathsf{E}\diamondsuit\gamma_{\text{ind}} & \text{a E sometime clause}
\end{array}
$$

∎

The intuition behind this formulation is that initial clauses provide the initial conditions for the computation while each of the step clauses represents a constraint upon the future behaviour of the formula, given the current conjunction of literals. Note that the $\mathsf{A}\square$ constraint is only utilised BNF as the one surrounding the conjunction of clauses which gives the clauses the 'universal' interpretation by propagating information they represent to each state along each path of the tree structure. At the same time, the $\mathsf{E}\square$ constraint is not utilised in the BNF, although, obviously it can be introduced based on $\mathsf{E}\square\alpha =_{\mathsf{def}} \mathsf{E}\neg\diamondsuit\neg\alpha$. Finally note that in the eventuality clauses the argument of the $\diamondsuit$ operator is a literal. This requirement is due to the potential application of the clausal resolution method to the specifications written in BNF. The clausal resolution method resolves (if possible) an eventuality, $\diamondsuit l$ in the scope of a path quantifier with the loop in l which is defined on all or some dedicated paths, see details in [2, 4].

3.2 Interpretation of BNF

For the interpretation of BNF clauses, utilising the notation of [22], we introduce an indexed tree-like model. Given *IND* is a countable set of indices, S is a set of states, $E \subseteq S \times S$ is a total binary relation over S, and L is an interpretation function $S \longrightarrow 2^{Prop}$, which maps a state $s_i \in S$ to the set of atomic propositions that are true at s_i. Then an indexed model structure $\mathscr{M} = \langle S, R, L, \overrightarrow{\text{ind}}, s_0 \rangle$ where $s_0 \in S$, and $\overrightarrow{\text{ind}} \subseteq R$ such that it is the argument of the mapping of every index ind $\in$ *IND* to the successor function $\overrightarrow{\text{ind}}$ such that $\overrightarrow{\text{ind}}$ is a total functional relation on S

It is easy to see that the underlying tree model above is an ω-tree satisfying the conditions of Definition 5.

A state $s_j \in S$ is an ind-successor state of state $s_i \in S \Leftrightarrow (s_i, s_j) \in \overrightarrow{\text{ind}}$. An infinite path $\chi_{s_i}^{\text{ind}}$ is an infinite sequence of states $s_i, s_{i+1}, s_{i+2}, \ldots$ such that for every j $(i \leq j)$, we have that $(s_j, s_{j+1}) \in \overrightarrow{\text{ind}}$.

In Figure 1 we define the relation '$\models$', omitting cases for Booleans, $\text{\textsc{t}}$ and $\text{\textsc{f}}$. Also recall that the basic modality $\mathsf{E}\square$ is not used in the BNF while $\mathsf{A}\square$ only appears as an outer modality preceeding the conjunction of the BNF clauses.

$$
\begin{array}{llll}
\langle \mathscr{M}, s_i \rangle & \models & \mathbf{start} & \Leftrightarrow \quad i = 0 \\
\langle \mathscr{M}, s_i \rangle & \models & \mathsf{A}\bigcirc B & \Leftrightarrow \quad \textit{for each } \mathsf{ind} \in \textit{IND and each } s' \in S, \textit{if } (s_i, s') \in \overrightarrow{\mathsf{ind}} \textit{ then } \langle \mathscr{M}, s' \rangle \models B \\
\langle \mathscr{M}, s_i \rangle & \models & \mathsf{E}\bigcirc B_{\mathsf{ind}} & \Leftrightarrow \quad \textit{there exists } s' \in S, \textit{such that } (s_i, s') \in \overrightarrow{\mathsf{ind}} \textit{ and } \langle \mathscr{M}, s' \rangle \models B \\
\langle \mathscr{M}, s_i \rangle & \models & \mathsf{A}\square B & \Leftrightarrow \quad \textit{for each } \chi_{s_i} \in \Pi_{\mathscr{M}} \textit{ and each } s_j \in \chi_{s_i}, \textit{ if } (i \leq j) \textit{ then } \langle \mathscr{M}, s_j \rangle \models B \\
\langle \mathscr{M}, s_i \rangle & \models & \mathsf{A}\Diamond B & \Leftrightarrow \quad \textit{for each } \chi_{s_i} \in \Pi_{\mathscr{M}}, \textit{there exists } s_j \in S, \textit{ such that } (i \leq j) \\
& & & \qquad \textit{and } \langle \mathscr{M}, s_j \rangle \models B \\
\langle \mathscr{M}, s_i \rangle & \models & \mathsf{E}\Diamond B_{\mathsf{ind}} & \Leftrightarrow \quad \textit{there exist } \chi_{s_i}^{\mathsf{ind}} \in \Pi_{\mathscr{M}} \textit{ and } s_j \in \chi_{s_i}^{\mathsf{ind}}, \textit{ such that } i \leq j, \langle \mathscr{M}, s_j \rangle \models B
\end{array}
$$

Figure 1: BNF semantics

Definition 7 *[Satisfiability, Validity] If $\mathscr{C}$ is in* BNF *then $\mathscr{C}$ is satisfiable if, and only if, there exists a model $\mathscr{M}$ such that $\langle \mathscr{M}, s_0 \rangle \models \mathscr{C}$. $\mathscr{C}$ is valid if, and only if, it is satisfied in every possible model.* ∎

3.3 BNF **examples.**

Here we give an example of the normal form and its informal interpretation. Noting that an initial BNF clause, $\mathbf{start} \Rightarrow F$, is understood as "*F* is satisfied at the initial state of some model $\mathscr{M}$"and any other BNF clause is interpreted taking also into account that it occurs in the scope of $\mathsf{A}\square$ let us consider a clause $\mathsf{A}\square(x \Rightarrow \mathsf{E}\bigcirc q_{\mathsf{ind}})$. It is understood as "*for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$, if x is satisfied at a state s_i then q must be satisfied at the moment, next to s_i, along some path associated with* ind *which departs from s_i*".

The clause $\mathsf{A}\square(x \Rightarrow \mathsf{E}\Diamond p_{\mathsf{ind}})$ has the following meaning "*for any fullpath χ and any state $s_i \in \chi$ $(0 \leq i)$, if x is satisfied at a state s_i then p must be satisfied at some state, say s_j $(i \leq j)$, along some path α_{s_i} associated with the limit closure[2] of* ind *which departs from s_i*'.

4 Büchi Tree automata

Definition 8 *A Büchi automaton, $\mathscr{B}$, on an infinite tree, T, is a tuple $\mathscr{B} = \langle \Sigma, D, S, \delta, F_0, F_B \rangle$ where: Σ is a finite alphabet; D is a finite set of branching degrees; $S = \{s_0, s_1, \ldots s_k\}$ is a finite set of states; $F_0 \subseteq S$ is a set of initial states; δ is a non-deterministic transition function satisfying; $\delta(s, \sigma, d) \subseteq S^d$, for every $s \in S, \sigma \in \Sigma$ and $d \in D$, and $F_B \subseteq S$ is a set of accepting states.* ∎

The transition function $\delta(s, \sigma, d)$ is the set of all tuples of states to which the automaton may evolve from state $s \in S$ of the arity $d \in D$ when it reads the symbol $\sigma \in \Sigma$.

A *run*, $\tau_{\mathscr{B}} : T \longrightarrow S$, of a Büchi tree automaton $\mathscr{B}$ over the input Σ-labelled tree (T, L) is an S labelled tree such that the root is labelled by a member of F_0 and the transitions conform with the transition function δ. Namely, visiting a state $s_i \in S$ with the branching degree d and reading $\sigma \in \Sigma$, an automaton makes a non-deterministic choice of a tuple $s_0, \ldots s_d \in \delta(s, \sigma, d)$, $1 \leq d \leq k$, makes d copies of itself and moves to the node $s_i \cdot s_j$ $(1 \leq j \leq d)$.

---

[2]Observe that limit closure has the properties of the reflexive transitive closure.

A run, $\tau_{\mathscr{B}}$, is *successful* if for every infinite branch of $\tau_{\mathscr{B}}$, there is an accepting state $s \in F_B$ that occurs infinitely often in this branch. An automaton $\mathscr{B}$ accepts the infinite tree T (in other terms, the language recognised by $\mathscr{B}$ is *not empty*) if it has a successful run $\tau_{\mathscr{B}}$.

5 From Büchi Tree automata to BNF

Here, given a Büchi tree automaton $\mathscr{B}$, we construct its characteristic formula, $\mathsf{BNF}_{\mathscr{B}}$, as a set of BNF clauses, following the main stages similar to those in [7]: encoding of the set of the initial states, representing the run of the automaton, the labelling, and the acceptance condition.

Let $\mathscr{B} = \langle \Sigma, D, S, \delta, F_0, F_B \rangle$ be a Büchi tree automaton with the states labelled as follows. For every proposition $p \in Prop$, for $\sigma \in \Sigma$, given $s_i \cdot s_k \in \delta(s_i, \sigma, d)$ (for $1 \le k \le d$), if $p \in \sigma$ then $p \in L(s_i \cdot s_k)$ else if $p \notin \sigma$ then $\neg p \in L(s_i \cdot s_k)$. In our translation we will explicitly encode this labelling. Now, we introduce formulae (1)–(4), which represent the main stages of the translation. Note that to simplify reading we will omit writing the outer $\mathsf{A}\,\square$, and will write a set of $\mathsf{BNF}_{\mathscr{B}}$ clauses rather than their conjunction; in some cases we will also give the formulation of the components of the translation not in the exact form of $\mathsf{BNF}_{\mathscr{B}}$ clauses to make the idea behind the translation more explicit. Each of these cases, where we need further simple manipulations to obtain the required BNF form, will be marked with $\{\star\}$.

Let $q_0, \ldots, q_l$ be new propositions of our BNF language such that q_i $(0 \le i \le l)$ encodes the state $s_i \in S$ of the automaton. Given that $q_0, \ldots, q_m$, $(0 \le m \le l)$ encode the initial states, F_0, of the automaton, we specify F_0 by the following BNF:

$$
\mathsf{BNF}_{init_{\mathscr{B}}} : \quad
\begin{array}{ll}
(1.1) & \textbf{start} \;\Rightarrow\; q_0 \vee \ldots \vee q_m \\[4pt]
(1.2) & \textbf{start} \;\Rightarrow\; \neg q_i \vee \bigwedge_{i \neq j}(\neg q_j) \quad \{\star\}
\end{array}
\tag{1}
$$

where $0 \le i \le m$, $0 \le j \le m$. From (1) it follows that the automaton can be at only one initial state at the first moment of time. Note that constraint (1.2) of the $\mathsf{BNF}_{init_{\mathscr{B}}}$ marked with '$\star$' should be further translated to the form required by the BNF.

Next, the transition function of the automaton is represented as follows.

Given the automaton $\mathscr{B}$ with the set of branching degrees $D = \{1, \ldots, d\}$, we associate with each element of the latter a set of new indices $IND = ind_1, \ldots ind_d$ used to label BNF clauses. Thus, when the automaton makes its d copies visiting a state $s_i \in S$, with the branching factor d, with each such successor node s_n of s_i we associate an index ind_j, $1 \le j \le d$.

$$
\mathsf{BNF}_{tran_{\mathscr{B}}} : \quad (2.1)\, q_i \Rightarrow \mathsf{E}\bigcirc q_{n_{ind_j}}
\tag{2}
$$

Thus each of d clauses in (2) reflects a successor node of s_i along the path labeled by ind_j. Note that unlike Büchi word automaton which can only be at one state at any particular time, a tree automaton makes copies choosing a corresponding tuple of states. This is exactly what we have represented above. Thus, for each q_i of branching degree d there are exactly d clauses of the form (2).

Next, we represent the unique labelling of the states of the automaton by the following set of clauses,

constructed for every q_i and every $x_i \in L(s_i)$.

$$BNF_{lab_{\mathscr{B}}}: \quad (3.1) \quad \textbf{start} \;\Rightarrow\; \neg q_i \vee \left[\left(\bigwedge_{x_i \in L(s_i)} x_i \right) \wedge \left(\bigwedge_{x_j \notin L(s_i)} \neg x_j \right) \right] \quad \{\star\} \qquad (3)$$

$$(3.2) \quad \textbf{T} \;\Rightarrow\; \mathsf{A}\bigcirc(\neg q_i \vee v)$$

$$((3.3) \quad v \;\Rightarrow\; \left[\left(\bigwedge_{x_i \in L(s_i)} x_i \right) \wedge \left(\bigwedge_{x_j \notin L(s_i)} \neg x_j \right) \right] \qquad \{\star\}$$

The Büchi acceptance condition is given by the following set of BNF clauses constructed for each $ind_i \in IND$.

$$
\begin{aligned}
BNF_{acc_{\mathscr{B}}}: \quad &(4.1) \quad \textbf{start} \;\Rightarrow\; y \qquad\qquad\qquad\qquad (4)\\
&(4.2) \quad\quad y \;\Rightarrow\; \mathsf{A}\square u \qquad \{\star\}\\
&(4.3) \quad\quad u \;\Rightarrow\; \mathsf{E}\lozenge l_{ind_i}\\
&(4.4) \quad\quad l \;\Rightarrow\; \mathsf{E}\bigcirc w_{ind)i}\\
&(4.5) \quad\quad w \;\Rightarrow\; \mathsf{E}\lozenge l_{ind)i}\\
&(4.6) \quad \textbf{start} \;\Rightarrow\; \neg l \vee q_n \vee \ldots \vee q_r\\
&(4.7) \quad\quad \textbf{T} \;\Rightarrow\; \mathsf{A}\bigcirc(\neg l \vee q_n \vee \ldots \vee q_r)
\end{aligned}
$$

where

- $q_n \ldots q_r$ encode the accepting states of the automaton and y, w, l and z are new propositions. This condition ensures that every path of the run hits an accepting state from F_B infinitely often. The use of the indices guarantees that we are staying in the 'context of a chosen path' when verifying the acceptance condition.

- Constraint 2 of the $BNF_{acc_{\mathscr{B}}}$ marked with '$\star$' is the abbreviation of the 'loop' in u. This loop in the BNF is represented by the following two clauses (i) $y \Rightarrow (z \wedge u)$, (ii) $z \Rightarrow \mathsf{A}\bigcirc(z \wedge u)$ (recall that all clauses are in the scope of the $\mathsf{A}\square$): from (i) we know that u is true at the state where y is true, and also that z is also true at that state, while from (ii) we derive that the every successor state should satisfy both u and z. This second clause (ii) ensures the recurrent presence of z in every subsequent state along every path, which, in turn, ensures that each subsequent state along every path also satisfies u.

Finally, let $\mathsf{BNF}'_{init_{\mathscr{B}}}$, $\mathsf{BNF}'_{tran_{\mathscr{B}}}$, $\mathsf{BNF}_{lab_{\mathscr{B}}}$ and $\mathsf{BNF}_{acc^B_{\mathscr{B}}}$ be obtained from $\mathsf{BNF}_{init_{\mathscr{B}}}$, $\mathsf{BNF}_{tran_{\mathscr{B}}}$, $\mathsf{BNF}_{lab_{\mathscr{B}}}$ and $\mathsf{BNF}_{acc^B_{\mathscr{B}}}$, respectively, by translating their components into the required form of BNF clauses. Now, a Büchi tree automaton $\mathscr{B}$ is characterized by the following BNF expression known as a *characteristic clause set* and abbreviated by $\mathsf{BNF}_{\mathscr{B}}$:

$$\mathsf{BNF}'_{init_{\mathscr{B}}} \wedge \mathsf{BNF}'_{tran_{\mathscr{B}}} \wedge \mathsf{BNF}_{lab_{\mathscr{B}}} \wedge \mathsf{BNF}'_{acc^B_{\mathscr{B}}} \qquad (5)$$

5.1 Correctness

Theorem 1 *Given a Büchi tree automaton $\mathscr{B}$, we can construct a characteristic clause set, $\mathsf{BNF}_{\mathscr{B}}$, such that $\mathscr{B}$ has an accepting run, $\tau_{\mathscr{B}}$ (over an infinite tree T), if and only if, $\mathsf{BNF}_{\mathscr{B}}$ is satisfiable.*

Proof.

(I) Left to right direction. The proof effectively follows the labelling chosen for BNF clauses described above. Given a Büchi tree automaton, $\mathscr{B} = \langle \Sigma, D, S, \delta, F_0, F_B \rangle$, on an infinite tree, recall that its accepting run $\tau_{\mathscr{B}}$ is an S-labelled tree (T, L) such that its root is labelled by a member of F_0 and the transitions conform with the transition function δ.

STEP 1. First, let $|IND| = d$, where $d \in D$ is the largest branching factor, and let the states of S be obtained from the corresponding nodes of T provided the states of a Büchi automaton are labelled as above, namely, such that for every proposition $p \in Prop$, for $\sigma \in \Sigma$, given $s_i \cdot s_k \in \delta(s_i, \sigma, d)$ (for $1 \leq k \leq d$), if $p \in \sigma$ then $p \in L(s_i \cdot s_k)$ else if $p \notin \sigma$ then $\neg p \in L(s_i \cdot s_k)$. This labelling guarantees that the mapping $\overrightarrow{ind}$ establishes the desired order over the states of S so every state $x \cdot k$ with the degree d ($1 \leq k \leq d$) of the tree model, is identified with the S-label of the node $x \cdot k$ of the run $\tau_{\mathscr{A}}$. Thus, the way how the labelling chosen for the BNF clauses guarantees that this tree becomes an underlying tree structure for the model $\mathscr{M} = \langle S, R, L, \overrightarrow{ind}, s_0 \rangle$ such that for every $p_k \in L(x \cdot k)$, $(\mathscr{M}, s \cdot k) \models p_k$.

STEP 2. Let a model $\mathscr{M}' = \langle S', R, L', \overrightarrow{ind}, s_0' \rangle$ be the same as $\mathscr{M}$ except for the interpretation of the new propositions $q_0, q_a, \ldots q_b, q_n, q_r, y, l, u, v, w$ and $z_1 \ldots z_m$ which we chose to satisfy at the appropriate states of S'. For example, $\langle \mathscr{M}', s_0 \rangle \models y$ and for every $s_n \neq s_0$, $\langle \mathscr{M}', s_n \rangle \not\models y$. Updating this way the model $\mathscr{M}$ into $\mathscr{M}'$ we guarantee that each component of the characteristic clause set is satisfied in $\mathscr{M}'$

(II). Right to left direction. We prove this by contradiction, i.e. assuming that the given Büchi tree automaton,

$$\mathscr{B} = \langle \Sigma, D, S, \delta, F_0, F_B \rangle$$

on an infinite tree, does not have an accepting run, we show that we cannot build a model for the characteristic clause set, $\mathrm{BNF}_{\mathscr{B}}$, for $\mathscr{B}$. The emptiness of the automaton would mean that the acceptance condition is violated. Thus, as there is no succefull run of the automaton, it should have an infinite branch χ, such that there is no accepting state $s_i \in F_B$ which occurs in χ infinitely often. According to the construction of the intermediate model structure for every $p_k \in L(x \cdot k)$, $(\mathscr{M}, s \cdot k) \models p_k$. Any model $\mathscr{M}' = \langle T', \leq, I' \rangle$ which agrees with $\mathscr{M}$ everywhere except for the interpretation of the new propositions appeared in the characteristic clause set, does not satisfy the latter. Indeed, since $q_n, \ldots, q_r$ are labels of the accepting states, and none of the accepting states occurs in χ infinitely often, for every q_j ($n \leq j \leq r$), for any $\mathscr{M}'$ we have that $\neg q_j$ becomes eventually always true along χ, and hence, in $\mathscr{M}'$ the conjunction $\neg q_n \wedge \ldots \wedge \neg q_r$ becomes always true from some point on along this path χ. This contradicts the satisfiability conditions for $\mathrm{BNF}'_{\mathscr{B}}$: because the conjunction $\neg q_n \wedge \ldots \wedge \neg q_r$ becomes always true from some point on, say $s_m \in S$ along χ, given also (4.6) and (4.7) we must have a loop in $\neg l$, from s_m, i.e. $\square \neg l$ becomes true along χ from s_m. As (4.3) and (4.5) are constructed for each $ind_i \in IND$, we would have a contradiction between this loop in $\neg l$ along χ and the request to fulfil the eventuality l along this path. ∎

6 From BNF to Büchi Tree automata

In this section we show how to effectively construct a Büchi Tree automaton from a given set $\mathscr{C}$ of BNF clauses. First, let us distinguish among BNF clauses the initial clauses and the global (step and sometime) clauses. The ideology is as follows. First we apply the augmentation technique developed for the clausal resolution for CTL [2] deriving $\mathscr{C}_{Aug}$ an augmented BNF. Then we show how to construct a model for $\mathscr{C}_{Aug}$. This technique involves a construction of a tableau as a labelled finite directed graph. The states of the graph are labelled by the sets of subformulae of $\mathscr{C}_{Aug}$. Let $Prop(\mathscr{C}_{Aug})$ be a set of all (different) propositions that occur within the clauses of $\mathscr{C}_{Aug}$. The important features of the underlying

transition function for the construction of the tableau is that the formula $A \Box \phi$, where ϕ is a conjunction of all global clauses of $\mathscr{C}_{Aug}$, occurs within each state. Thus, global clauses play the role of a guide for the transitions. A transition from a state s_i to a state s_j is provided if a label for s_j is consistent.

Once a tableau $\mathscr{G}_{\mathscr{C}_{Aug}}$ is constructed, labels of some states might contain formulae of the type $P\Diamond l$. Thus, we check if such an eventuality is satisfied. During this procedure we delete those states (and their successors) of a graph $\mathscr{G}_{\mathscr{C}_{Aug}}$ which contain unsatisfied eventualities. As some states of a graph now might be without any successor, we will delete such states. The resulting graph, $\mathscr{R}$, is called a *reduced tableau*.

It has been shown that the set of BNF clauses is unsatisfiable, if, and only if, its reduced tableau is empty [2].

Definition 9 (Augmentation) *Given a set of* BNF *clauses,* $\mathscr{C}$*, we construct an* **augmented** *set* $\mathscr{C}_{Aug}$ *as follows.*

1. *Create a list* $\{EVEN\} = E\Diamond l_1, E\Diamond l_2, \ldots, \ldots E\Diamond l_n, A\Diamond l_{n+1}, A\Diamond l_{n+2}, \ldots, A\Diamond l_m$ $(0 \leq n \leq m)$ *of all different eventualities contained in* $\mathscr{C}$*.*

2. *To keep track of the eventualities, create a list* $W = w_1, w_2, \ldots, w_m$ *of new propositions (that do not occur within clauses of* $\mathscr{C}$*) such that each* $w_i \in W$ *is associated with the i-th eventuality within EVEN.*

3. *For every sometime clause* $C \Rightarrow P\Diamond l_{i \mathrm{ind}_i}$*, to guarantee the correspondence between* w_i *and* l_i*, add the corresponding set of formulae, defined below.*

   3a If $P\Diamond l_i = A\Diamond l_i$*, then add to the set of* BNF *clauses* $\mathscr{C}$*, the following formulae:*

$$
\begin{aligned}
\mathbf{start} &\Rightarrow \neg C \vee l_i \vee w_i \\
w_i &\Rightarrow A\bigcirc(l_i \vee w_i) \\
\mathbf{T} &\Rightarrow A\bigcirc(\neg C \vee l_i \vee w_i).
\end{aligned}
\tag{6}
$$

   3b If $P\Diamond l_i = E\Diamond l_{i\mathrm{ind}}$*, then add to the set of* BNF *clauses* $\mathscr{C}$*, the following formulae:*

$$
\begin{aligned}
\mathbf{start} &\Rightarrow \neg C \vee l_i \vee w_i \\
w_i &\Rightarrow E\bigcirc(l_i \vee w_i)_{\mathrm{ind}} \\
\mathbf{T} &\Rightarrow E\bigcirc(\neg C \vee l_i \vee w_i)_{\mathrm{ind}}
\end{aligned}
\tag{7}
$$

∎

Let $\mathscr{C}_{Aug}$ be an augmented set of BNF clauses. Abbreviating by *In* the conjunction of the right hand sides of the initial clauses of $\mathscr{C}_{Aug}$ and by ϕ the conjunction of its step and eventuality clauses, we can represent the set, $\mathscr{C}_{Aug}$ as a formula $In \wedge A \Box \phi$. It is easy to show from the BNF semantics that the following holds: $\langle \mathscr{M}, s_0 \rangle \models \mathscr{C} \Leftrightarrow \langle \mathscr{M}, s_0 \rangle \models In \wedge A \Box \phi$.

Let an *elementary* formula be either a literal, or has its main connective as $P\bigcirc$. Each *non-elementary* formula is further classified as either a conjunctive, α-formula, or a disjunctive, β-formula.

A basic BNF modality now is qualified according to its fixpoint definition (in the equations below μ and ν stand for 'minimal fixpoint' and 'maximal fixpoint' operators, respectively)

$$
\begin{aligned}
A\Box\varphi &= \nu\rho(\varphi \wedge A\bigcirc\rho) \\
E\Diamond\varphi &= \mu\rho(\varphi \vee E\bigcirc\rho) \\
A\Diamond\varphi &= \mu\rho(\varphi \vee A\bigcirc\rho)
\end{aligned}
\tag{8}
$$

The only BNF modality that occurs within BNF clauses as a maximal fixpoint is $A \Box \phi$ and in our representation of a set of BNF clauses as $In \wedge A \Box \phi$, it has the only occurrence as the main connective in $A \Box \phi$, hence, the following α-expansion rules will be appropriate:

α	α_1	α_2
$B \wedge C$	B	C
$A\,\square B$	B	$A\bigcirc A\,\square B$
$\neg(B \vee C)$	$\neg B$	$\neg C$

(9)

BNF modalities understood as a minimal fixpoints are $A\diamondsuit\phi$ and $E\diamondsuit\phi$, hence, the following β-expansion rules will be appropriate:

β	β_1	β_2
$B \Rightarrow C$	$\neg B$	C
$B \vee C$	B	C
$A\diamondsuit B$	B	$A\bigcirc A\diamondsuit B$
$E\diamondsuit B_{\text{ind}}$	B	$E\bigcirc E\diamondsuit B_{\text{ind}}$

(10)

Here the last two constraints must be further transformed into the appropriate structure of BNF clauses, however, the current representation is preferable as it illustrates the intuition. Let $Even_{\mathscr{C}_{Aug}}$ be a list of eventualities as defined in Definition 9 and let $Prop_{\mathscr{C}_{Aug}}$ be a set of all (different) propositions that occur within the clauses of $\mathscr{C}_{Aug}$. By an evaluation of a proposition $p_i \in Prop_{\mathscr{C}_{Aug}}$ we understand the function $Prop_{\mathscr{C}_{Aug}} \longrightarrow \{0,1\}$. Now, let $Val(Prop_{\mathscr{C}_{Aug}})$ be a set of all possible evaluations of the elements of the $Prop_{\mathscr{C}_{Aug}}$. Finally, let $D = \text{ind}_1, \text{ind}_2 \ldots \text{ind}_k$ be a list of all different indices that occur in $\mathscr{C}$.

We adapt the notion of the generalized Fischer-Ladner closure [16] introduced for CTL formulae in [13] for our case of BNF.

Definition 10 (Generalized Fischer-Ladner closure for BNF**)** *Let $\mathscr{C}$ be a set of* BNF *clauses, let $In \wedge A\,\square\phi$ be its equivalent formula, where 'In' abbreviates the conjunction of the right hand sides of the initial clauses of $\mathscr{C}$ and ϕ abbreviates the conjunction of the global clauses within $\mathscr{C}$. Then the least set of formulae which contains $\mathscr{C}$ and satisfies the conditions below is the generalised Fischer-Ladner closure of $\mathscr{C}$, abbreviated by $GFL(\mathscr{C})$.*

  (GFL1) $In \wedge A\,\square\phi$ is an element of $GFL(\mathscr{C})$.
  (GFL2) If B is an element of $GFL(\mathscr{C})$ then any subformula of B is an element of $GFL(\mathscr{C})$.
  (GFL3) If $A\,\square B \in GFL(\mathscr{C})$ then $A\bigcirc A\,\square B \in GFL(\mathscr{C}))$.
  (GFL4) If $E\diamondsuit B_{\text{ind}} \in GFL(\mathscr{C})$ then $E\bigcirc E\diamondsuit B_{\text{ind}} \in GFL(\mathscr{C})$.
  (GFL5) If $A\diamondsuit B \in GFL(G)$ then $A\bigcirc A\diamondsuit B \in GFL(\mathscr{C})$.
  (GFL6) If $B \in GFL(G)$ and B is not of the form $\neg C$ then $\neg B \in GFL(\mathscr{C}))$. ∎

Now given a set of BNF clauses represented by $In \wedge A\,\square\phi$, we construct a labelled finite graph $\mathscr{G}$ *incrementally* as follows

1. The initial state is labelled by $In \wedge A\,\square\phi$.

2. Inductively assume that a graph has been constructed with nodes labelled by the subsets of $GFL(In \wedge A\,\square\phi)$, where some formulae during such construction are marked.

   Note: when providing the transitions from a state n if nodes t_1 and t_2 have the same label and the same marked formulae, they are identified, and we delete one of them, say t_2, drawing an edge to t_1. Also, if a transition leads to a node which does not satisfy the propositional consistency criteria, we delete this node.

   Given a node n with the label Γ, choose an unmarked formula, say B, and apply an appropriate expansion rule as follows:

2a. If B is an α-formula, then create a successor of n labelled by
$\Gamma \cup \{\alpha_1, \alpha_2\}$ and mark B in the label.

2b. If B is a β-formula, then create two successors of n and label one of them by $\Gamma \cup \{\beta_1\}$ while another one by $\Gamma \cup \{\beta_2\}$ and mark B in the label.

3. If all non-elementary formulae within a node are marked, such a node is called a *state*. Let s, be a state whose label contains the following *next*-time formulae:

$$\mathsf{A}\bigcirc B_1 \ldots \mathsf{A}\bigcirc B_k, \mathsf{E}\bigcirc C_{1_{\mathsf{ind}_1}} \ldots \mathsf{E}\bigcirc C_{r_{\mathsf{ind}_r}}.$$

Merge all C_i $(1 < i \le q)$ which have identical indices, (for example, given $\mathsf{E}\bigcirc p_{\mathsf{f}}$ and $\mathsf{E}\bigcirc q_{\mathsf{f}}$, producing $\mathsf{E}\bigcirc(p \wedge q)_{\mathsf{f}})$ obtaining in this way

$$\mathsf{A}\bigcirc B_1 \ldots \mathsf{A}\bigcirc B_k, \mathsf{E}\bigcirc C_{1_{\mathsf{ind}_1}} \ldots \mathsf{E}\bigcirc C_{m_{\mathsf{ind}_m}}.$$

Then create the successors $d_1 \ldots d_m$ of s labelled respectively by

$$\{B_1 \ldots B_k, C_1\} \ldots \{B_1 \ldots B_k, C_m\}.$$

4. Repeat steps 2 and 3 until no more new nodes are generated.

Now a tableau is a structure $\mathscr{G}_{\mathscr{C}_{Aug}} = (N, E, L)$, which satisfies the following conditions

- N is a set of those nodes that are states in the construction above,

- E is a set of edges such that for $s, t \in N$, $E(s,t)$ if, and only if, t is an immediate successor of s, and

- L is a set of labels, so for $s \in N$, the label of s is $L(s)$.

In the following we will utilise the concept of pseudofulfillment of eventualities, adapting a general definition to our case.

Definition 11 (Pseudo fulfilled eventuality) *Given a tableau $\mathscr{C}_{Aug}$, if a state s contains an eventuality* $\mathbf{P}\diamondsuit l$*, then* $\mathbf{P}\diamondsuit l$ *is pseudo fulfilled, if* $\mathscr{G}_R$ *satisfies the following conditions.*

- *If* $\mathbf{P}\diamondsuit l = \mathsf{A}\diamondsuit l$ *then there exists a finite subgraph H of $\mathscr{G}_R$ with s as its root, such that for any terminal state $t \in H$, $l \in t$.*

- *If* $\mathbf{P}\diamondsuit l = \mathsf{E}\diamondsuit l_{\mathsf{f}}$ *then there exists a finite subgraph H of $\mathscr{G}_R$ with s as its root, such that H has a finite path π_s which departs from s and satisfies the following condition: each state $t_{i+1} \in \pi_s$, is the $f-th$ successor of t_i, and l is satisfied at the terminal state of π_s.* ∎

Note that pseudo fulfilment for $\mathsf{E}\diamondsuit l_{\mathsf{ind}}$ means that we have a path which contains a node satisfying l where every successor node is an ind-successor of the previous one, i.e. the one which is the successor node along the ind path.

Given a tableau $\mathscr{G}_{\mathscr{C}_{Aug}}$, apply the following deletion rules. If a state has no successors, then delete this state and all edges leading to it. If a state contains an eventuality which is not pseudo fulfilled, delete this state.

Finally, if a state contains $\mathsf{E}\bigcirc C_{\mathsf{ind}}$ and does not have an ind-successor which contains C, then delete this state. The resulting graph is called the reduced tableau.

Theorem 2 *([2]) For any* BNF *set $\mathscr{C}$, its reduced tableau is empty, if and only if, $\mathscr{C}$ is unsatisfiable.*

Now from a non-empty reduced graph $\mathcal{G}_{\mathcal{C}_{Aug}} = (N, E, L)$ for an augmented BNF we can construct a Büchi Tree Automaton, $\mathcal{B} = \langle \Sigma, D, S, \delta, F_0, F_B \rangle$ following the standard technique, for example, [21].

- $\Sigma = 2^{Prop_{\mathcal{C}_{Aug}}}$, where $Prop_{\mathcal{C}_{Aug}}$ is a set of propositions of the clause set $\mathcal{C}_{Aug}$;

- D is the set of indices as defined for the construction of the Generalized Fischer-Ladner closure in Def. 10;

- S is a finite set, N, of states of $\mathcal{C}_{Aug}$;

- δ is a transition function which corresponds to the edge relation E of the reduced tableau;

- F_0 is a set of those states that satisfy all the initial clauses occurring within the clause set $\mathcal{C}_{Aug}$;

- $F_B = S = N$ is a set of all states in the reduced tableau.

Theorem 3 *Given a set, $\mathcal{C}$, of BNF clauses, we can construct a Büchi tree automaton $\mathcal{B}$ such that $\mathcal{C}$ is satisfiable, if and only if, $\mathcal{B}$ has an accepting run, $\tau_{\mathcal{B}}$.*

Proof. According to Theorem 2, if a set, $\mathcal{C}$, of BNF clauses is satisfiable then so is the reduced tableau for $\mathcal{C}_{Aug}$. Due to the construction of the latter, following the transitions E, we can unwind it into an infinite tree T such that the root of the tree is labelled by the $\mathcal{C}_{Aug}$ and if a node $s_i \in N$ has ind-successor nodes they become ind-successors of the corresponding node x_i of T. The labelling of the states of N gives the labelling of Σ for the nodes of T. Now consider the Σ-labelled tree (T, Σ) as a tree over which the desired run τ of the automaton can be defined. The above construction of the automaton's accepting states guarantees that every path through the run would hit an accepting state infinitely often.

On the other hand, if we have an unsatisfiable set of BNF clauses then its reduced tableau is empty, and therefore, the automaton $\mathcal{B}$ whose construction is based upon the properties of this reduced tableau, cannot have an accepting run. ∎

7 Example

As an example of the syntactic representation of Büchi tree automaton in terms of BNF let us consider the following small automaton $\mathcal{B} = \langle \Sigma, D, S, \delta, F_0, F_B \rangle$ where: the alphabet $\Sigma = \{p, r\}$, the set of branching degrees is $D = \{d_1 = 2\}$; the set of states $S = \{s_0, s_1\}$ with the initial states $F_0 = s_0$ and the accepting state $F_B = s_1$. Finally, let the transition function be defined as follows:

$\delta(s_0, p, 2) = < s_0, s_0 >$, $\delta(s_1, p, 2) = < s_0, s_0 >$, $\delta(s_0, r, 2) = < s_1, s_1 >$, $\delta(s_1, r, 2) = < s_1, s_1 >$.

This is a non-empty automaton, and its accepting run hits an accepting state s_1 infinitely often on every branch. Now we will present the components of the BNF for this automaton. With q_1, q_2, for the encoding of the states of the automaton and selecting q_1 to encode the initial state, in equations (11)-(14) we give the components of the encoding of the given automaton, noting that v in the representation of the labelling, and y, l, u, w in the representations of the acceptance conditions are fresh variables. First, the encoding of the initial states of the given automaton is represented by the set of clauses (11).

$$\text{BNF}_{init_{\mathcal{B}}} \tag{11}$$
$$\text{1.} \quad \textbf{start} \Rightarrow q_1$$

Next, the transition function of the given automaton is represented by the set of clauses (12).

$$\text{BNF}_{tran_{\mathscr{B}}} \tag{12}$$

$$
\begin{aligned}
&12.1 \quad q_1 \Rightarrow \mathsf{E}\bigcirc q_{1\,ind_1}\\
&12.2 \quad q_1 \Rightarrow \mathsf{E}\bigcirc q_{2\,ind_2}\\
&12.3 \quad q_2 \Rightarrow \mathsf{E}\bigcirc q_{1\,ind_1}\\
&12.4 \quad q_2 \Rightarrow \mathsf{E}\bigcirc q_{2\,ind_2}
\end{aligned}
$$

The set of clauses (13) represent the labelling of the given automaton.

$$BNF_{lab_{\mathscr{B}}} \tag{13}$$

$$
\begin{aligned}
&13.1 \quad \textbf{start} \quad \Rightarrow \neg q_1 \vee p \wedge r\\
&13.2 \quad \textbf{start} \quad \Rightarrow \neg q_2 \vee p \wedge r\\
&13.3 \quad \textbf{\scriptsize T} \quad\;\; \Rightarrow \mathsf{A}\bigcirc(\neg q_1 \vee v)\\
&13.4 \quad v \quad\quad\; \Rightarrow p \wedge r\\
&13.5 \quad \textbf{\scriptsize T} \quad\;\; \Rightarrow \mathsf{A}\bigcirc(\neg q_2 \vee v)
\end{aligned}
$$

Finally, for the automaton acceptance conditions we have the set of clauses (14), where l, y, u, w are fresh variables.

$$BNF_{acc_{\mathscr{B}}} \tag{14}$$

$$
\begin{aligned}
&14.1 \quad \textbf{start} \quad \Rightarrow y\\
&14.2 \quad \textbf{start} \quad \Rightarrow \neg l \vee q_1\\
&14.3 \quad \textbf{\scriptsize T} \quad\;\; \Rightarrow \mathsf{A}\bigcirc(\neg l \vee q_1)\\
&14.4 \quad y \quad\quad\; \Rightarrow \mathsf{A}\square u\\
&14.5 \quad u \quad\quad\; \Rightarrow \mathsf{E}\Diamond l_{ind_1}\\
&14.6 \quad l \quad\quad\; \Rightarrow \mathsf{E}\bigcirc w_{ind_1}\\
&14.7 \quad w \quad\quad\; \Rightarrow \mathsf{E}\Diamond l_{ind_1}
\end{aligned}
$$

First, we show that the acceptance condition is properly simulated by the above BNF. Indeed, every state along every path of the underlying computation tree of the BNF, satisfies u, due to clauses 14.1 and 14.4. Now, pick an arbitrary state s_i along some fullpath, say ξ. Following 14.5, there should be a state s_j, $i \le j$, along the path $Suf(\xi, s_j)$, which is labeled ind_1 such that it satisfies l. Following 14.6, the ind-successor of the state s_j, say s_m, satisfies w. Hence by 14.7, there should be a state, say, s_k, $m \le k$ along the path $Suf(\xi, s_m)$ labeled by ind_1 such that s_k satisfies l. Due to 14.3, states s_j and s_k satisfy q_1. Repeating this chain of reasoning steps regarding the satisfiability of l we derive the recurrent satisfiability of q_1, which corresponds to the acceptance condition for this automaton, to hit the acceptance state s_1 infinitely often.

8   Discussion

We have shown that normal form used for the clausal resolution method for a variety of CTL-type logics is expressive enough to give the succinct high-level syntactic representation of a Büchi tree automaton. This represents a significant step in establishing the exact expressiveness of this formalism relating it to this important class of tree automata. As a consequence, based on the known expressiveness results, that Büchi tree automata are as expressive as CTL extended by the propositional quantification, we can also treat BNF as a kind of normal form for the latter: we can directly translate given problem specifications into BNF and apply as a verification method a *deductive reasoning technique* – the temporal resolution

technique. This paper justifies that BNF is suitable to reason about a wider range of branching-time specifications and is able to capture exactly those that are captured by Büchi tree automata.

Moreover, another very promising route is emerging here - tackling invariants. For example, invariants are extremely important in the emerging trend of developing complex but re-configurable software systems. Here one of the most challenging problems is assuring that invariants are maintained during system reconfiguration. In BNF we are enabled not only to represent some given, or explicit invariants, but also to discover implicit, hidden invariants. It has been shown how the clausal temporal resolution technique developed for temporal logic provides an effective method for searching for invariants in the linear time setting [8]. Our results enable the extension of this method to branching-time framework, and the detailed analysis of this route forms one direction of future research.

Also, in the future we will investigate the representation of alternating tree automata in BNF, where we expect results similar to the linear-time case [7].

References

[1] Artie Basukoski & Alexander Bolotov (2005): *Search Strategies for Resolution in CTL-Type Logics: Extension and Complexity*. In: *12th International Symposium on Temporal Representation and Reasoning (TIME 2005)*, pp. 195–197. Available at `http://dx.doi.org/10.1109/TIME.2005.32`.

[2] Alexander Bolotov (2000): *Clausal Resolution for Branching-Time Temporal Logic*. Ph.D. thesis, Department of Computing and Mathematics, The Manchester Metropolitan University.

[3] Alexander Bolotov & Artie Basukoski (2006): *A Clausal Resolution Method for Branching-Time Logic ECTL+*. Annals of Mathematics and Artificial Intelligence 46(3), p. 235–263, doi:10.1007/s10472-006-9018-1. Available at `https://doi.org/10.1007/s10472-006-9018-1`.

[4] Alexander Bolotov & Artie Basukoski (2006): *A Clausal Resolution Method for Extended Computation Tree Logic ECTL*. Journal of Applied Logic 4(2), pp. 141–167, doi:10.1016/j.jal.2005.06.003. Available at `https://doi.org/10.1016/j.jal.2005.06.003`.

[5] Alexander Bolotov & Clare Dixon (2000): *Resolution for Branching Time Temporal Logics: Applying the Temporal Resolution Rule*. In: *Seventh International Workshop on Temporal Representation and Reasoning, TIME 2000, Nova Scotia, Canada, July 7-9, 2000*, IEEE Computer Society, pp. 163–172, doi:10.1109/TIME.2000.856598. Available at `https://doi.org/10.1109/TIME.2000.856598`.

[6] Alexander Bolotov & Michael Fisher (1999): *A Clausal Resolution Method for CTL Branching-time Temporal Logic*. Journal of experimental and theoretical artificial intelligence 11(1), pp. 77–93, doi:10.1080/095281399146625. Available at `https://doi.org/10.1080/095281399146625`.

[7] Alexander Bolotov, Michael Fisher & Clare Dixon (2002): *On the Relationship between w-automata and Temporal Logic Normal Forms*. Journal of Logic and Computation 12(4), pp. 561–581. Available at `http://dx.doi.org/10.1093/logcom/12.4.561`.

[8] James Brotherston, Anatoli Degtyarev, Michael Fisher & Alexei Lisitsa (2002): *Searching for Invariants Using Temporal Resolution*. In Matthias Baaz & Andrei Voronkov, editors: *Logic for Programming, Artificial Intelligence, and Reasoning, 9th International Conference, LPAR 2002, Tbilisi, Georgia, October 14-18, 2002, Proceedings*, Lecture Notes in Computer Science 2514, Springer, pp. 86–101, doi:10.1007/3-540-36078-6_6. Available at `https://doi.org/10.1007/3-540-36078-6_6`.

[9] Julius Richard Büchi (1962): *On a Decision Method in Restricted Second-order Arithmetics*. In: *Proc.of International Congress of Logic, Methodology and Philosophy of Science*, Stanford University Press, pp. 1–12.

[10] Clare Dixon (2021): *Theorem Proving Using Clausal Resolution: From Past to Present*. In Paul C. Bell, Patrick Totzke & Igor Potapov, editors: *Reachability Problems - 15th International Conference, RP 2021,*

Liverpool, UK, October 25-27, 2021, Proceedings, Lecture Notes in Computer Science 13035, Springer, pp. 19–27, doi:10.1007/978-3-030-89716-1_2.

[11] Clare Dixon, Alexander Bolotov & Michael Fisher (2005): *Alternating Automata and Temporal Logic Normal Forms*. Annals of Pure and Applied Logic 135(1-3), pp. 263–285. Available at `http://dx.doi.org/10.1016/j.apal.2005.03.002`.

[12] E. Allen Emerson (1990): *Temporal and Modal Logic.* In J. van Leeuwen, editor: *Handbook of Theoretical Computer Science: Volume B, Formal Models and Semantics.*, Elsevier, pp. 996–1072.

[13] E. Allen Emerson & Joseph Y. Halpern (1985): *Decision procedures and expressiveness in the temporal logic of branching time.* Journal of Computer and System Sciences 30(1), pp. 1–24.

[14] E. Allen Emerson & Joseph Y. Halpern (1986): *"Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic*. Journal of ACM 33(1), p. 151–178, doi:10.1145/4904.4999. Available at `https://doi.org/10.1145/4904.4999`.

[15] E. Allen Emerson & A. Prasad Sistla (1984): *Deciding Branching Time Logic*. In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, Association for Computing Machinery, New York, NY, USA, p. 14–24, doi:10.1145/800057.808661. Available at `https://doi.org/10.1145/800057.808661`.

[16] Michael J. Fischer & Richard E. Ladner (1979): *Propositional dynamic logic of regular programs*. Journal of Computer and System Sciences 18(2), pp. 194–211, doi:https://doi.org/10.1016/0022-0000(79)90046-1.

[17] Michael Fisher (1997): *A Normal Form for Temporal Logics and its Applications in Theorem-Proving and Execution*. Journal of Logic and Computation 7(4), pp. 429–456, doi:10.1093/logcom/7.4.429. Available at `https://doi.org/10.1093/logcom/7.4.429`.

[18] Michael Fisher, Clare Dixon & Martin Peim (2001): *Clausal Temporal Resolution*. ACM Transactions on Computational Logic 2(1), pp. 12–56, doi:10.1145/371282.371311. Available at `https://doi.org/10.1145/371282.371311`.

[19] François Laroussinie & Nicolas Markey (2014): *Quantified CTL: Expressiveness and Complexity*. Logical Methods in Computer Science 10(4), pp. 1 – 45, doi:10.2168/LMCS-10(4:17)2014. Available at `https://doi.org/10.2168/LMCS-10(4:17)2014`.

[20] Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon & Michael Fisher (2019): *Formal Specification and Verification of Autonomous Robotic Systems: A Survey*. ACM Comput. Surv. 52(5), pp. 100:1–100:41, doi:10.1145/3342355. Available at `https://doi.org/10.1145/3342355`.

[21] Moshe Y. Vardi (2007): *Automata-theoretic Techniques for Temporal Reasoning*. In Patrick Blackburn, J. F. A. K. van Benthem & Frank Wolter, editors: *Handbook of Modal Logic*, Studies in logic and practical reasoning 3, North-Holland, pp. 971–989, doi:10.1016/s1570-2464(07)80020-6. Available at `https://doi.org/10.1016/s1570-2464(07)80020-6`.

[22] Lan Zhang, Ullrich Hustadt & Clare Dixon (2009): *A Refined Resolution Calculus for CTL*. In: *CADE-22*, pp. 245–260. Available at `http://dx.doi.org/10.1007/978-3-642-02959-2_20`.

[23] Lan Zhang, Ullrich Hustadt & Clare Dixon (2010): *CTL-RP: A computation Tree Logic Resolution Prover*. AI Commun. 23(2-3), pp. 111–136, doi:10.3233/AIC-2010-0463. Available at `https://doi.org/10.3233/AIC-2010-0463`.