Alternate Derivation of the Coalgebraic Representation of Büchi Automata Using Game Semantics

Research Internship Report

Jorrit de Boer

Abstract. *We provide a review of existing literature for describing Büchi automata coalgebraically using trace semantics. To do this we also explain the modal μ-calculus and a coalgebraic model of nondeterministic systems. Finally, we present an alternate derivation of the coalgebraic model for Büchi automata using game semantics, which gives a new conceptual perspective on the existing coalgebraic framework.*

1 Introduction

Büchi automata are crucial in theoretical computer science for modeling and verifying systems with infinite behaviors [3, 10]. Büchi automata are generally nondeterminstic, allowing them to capture uncertainty and multiple outcomes [7]. Because of their ability to capture nondeterminstism and handle infinite sequences of events, Büchi automata are crucial for verifying systems that run indefinitely, such as operating systems or network protocols.

Coalgebra provides an abstract framework for modeling state-based, dynamic systems [8]. Techniques such as *coinduction* allow for reasoning about infinite structures, while *bisimulation* offers a formal way to establish behavioral equivalence between systems. By modeling Büchi automata coalgebraically, these powerful tools can be applied for reasoning about infinite behaviors and nondeterminism.

The first goal of this report is to provide an understanding of the coalgebraic semantics using *trace semantics* of Büchi automata described in [9]. To do so we also explain the *modal μ-calculus* [1], a system for verifying properties of transition systems, and a coalgebraic model of nondeterministic systems from [4], upon which the construction for the Büchi automata builds. By outlining these concepts we advance our first goal of the research internship, which is to gain an understanding of the current research into this topic.

Secondly, we provide an alternate derivation of this coalgebraic representation using *game semantics* [3:Chapter 10]. Game semantics is a framework for describing a system in terms of a two-player game between a *verifier* and a *refuter* who want to verify, respectively refute, a statement. We interpret a system of equations occuring in the coalgebraic representation as a game, to which we then apply established theorems from game semantics to derive the coincidence between the coalgebraic model and the traces of the Büchi automaton. Our game theoretic proof contrasts the algebraic proof given in [9]. This formulation using game semantics might reveal connections to coalgebra automata which is based on game theoretic techniques [6].

The document is outlined as follows. In Section 2 we provide some background and relevant definitions for the rest of the report. In Section 3 we provide the coalgebraic representations of nondeterministic systems and Büchi automata from [4] and [9], respectively. In Section 4 we present our alternate derivation of the coincidence result given in the section before. Finally, in Section 5 we summarize the results and suggest directions for future work.

2 Background

2.1 Büchi Automata

Let us consider a very simple motivating example of a Büchi automaton, shown in Figure 1.

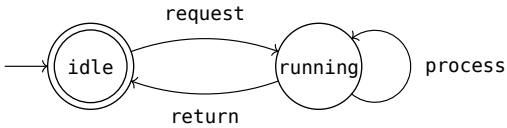

Figure 1: Example of a Büchi automaton.

This system represents some machine that takes requests, processes them, and returns some result. One might want to verify that this machine does not get stuck. In terms of the system shown, this would mean that the machine always ends up in the `idle` state again.

This behavior can be modeled using a Büchi automaton. A Büchi automaton, namely, is an automaton which models infinite behavior, and accepts those infinite words for which there is a path through the automaton where the transitions are labeled by the letters of the word, an there is an accepting state that the path moves through infinitely many times. In this example, we make the `idle` state accepting, so the automaton accepts those words that always take the `return` transition again, and thus do not process indefinitely.

We can now give a formal definition of a Büchi automaton, and its *accepted language*:

> **Definition 2.1**: A (nondeterministic) Büchi Automaton is a tuple $A = \langle S, \Sigma, \delta, s_0, F \rangle$, with S a finite set of states, Σ the finite alphabet, $s_0 \in S$ the initial state, $\delta : S \times \Sigma \to \mathcal{P}(S)$ the transition function, $F \subseteq S$ the set of *final* (or *accepting*) states.

A *run* of a Büchi Automaton A on an ω-word $w = \sigma_0\sigma_1 ... \in \Sigma^\omega$ is an infinite sequence of states $s_0, s_1, ... \in S^\omega$, such that s_0 is the initial state and for every $n \in \omega$, $s_{n+1} \in \delta(s_n, \sigma_n)$. A run is *accepting* if it passes through an accepting state infinitely many times. Equivalently (because F is finite), a run $\rho = s_0, s_1, ...$ is accepted if $\{i \mid s_i \in F\}$ is an infinite set. A word w is *accepted* by a Büchi automaton A if there is an acccepting run of A on w. Finally, the *accepted language* $L(A)$ of a Büchi automaton, is the set of words accepted by A, and $L(A)(s)$ is the set of words accepted by A starting in state s.

Indeed we now see that the accepted language for the example automaton is $(\texttt{request} \cdot \texttt{process}^* \cdot \texttt{return})^\omega$, where $*$ indicates repeating some set of letters/transitions some finite number of times (including zero) and ω indicates repeating indefinitely. That is, the machine gets a request, processes for at most some *finite* number of transitions and then returns some result. It does not get stuck processing indefinitely.

2.2 Parity Tree Automata

Büchi automata are actually a specific instance of *parity tree automata*. In this section we introduce this more general automaton. Although we mainly discuss Büchi automata in the report, we mention parity tree automata because the coincidence results presented in Section 3.2 also hold for parity tree automata, as discussed further in Section 3.2.

Instead of the acceptence criterion for Büchi automaton, we can use the parity acceptence condition. In this case, the states are not divided into accepting and non-accepting. Instead, every state has a priority, determined by $\Omega : S \to \omega$. A run $\rho = s_0, s_1, ...$ of an automaton A on a word w is then accepting if the maximum priority that occurs infinitely often is even. I.e., $\max\{\Omega(s) \mid s \text{ occurs infinitely often in } \rho\}$ is even. The Büchi acceptence criterion is the special case where non-accepting states have parity 1 and accepting states have parity 2.

Secondly, instead of words we can have a tree automaton. In this case the alphabet Σ is *ranked* and has an arity function function $|_| : \Sigma \to \omega$ indicating the number of successors a letter has. Let Tree_Σ be the set of trees whose nodes are labeled with letters $\sigma \in \Sigma$ and whose branching is consistent with the arity of the letters. For example, if $|\sigma| = 2$ for all $\sigma \in \Sigma$, a tree $\tau \in \mathrm{Tree}_\Sigma$ is a binary tree with labels $\sigma \in \Sigma$. If $|\sigma| = 1$ for all $\sigma \in \Sigma$, Tree_Σ is just the set of infinite words over Σ.

We can now define a parity tree automaton:

Definition 2.2: A (nondeterministic) Parity Tree Automaton is a tuple $A = \langle S, \Sigma, \delta, s_0, \Omega \rangle$, with S a finite set of states, Σ a ranked alphabet with arity function $|__| : \Sigma \to \omega$, $s_0 \in S$ the initial state, $\delta : S \times \Sigma \to \mathcal{P}(S^*)$ the transition function where for each $\sigma \in \Sigma$ if $|\sigma| = n$ then $\delta(s)(\sigma) \subseteq S^n$, and $\Omega : S \to \omega$ assigns a parity to each state.

A run ρ of the automaton A on a tree $\tau \in \mathrm{Tree}_\Sigma$ is the tree τ where the labels are replaced from letters $\sigma \in \Sigma$ to states $s \in S$ such that the root of the tree $\rho_0 = s_0$ is the initial state, and for a node in τ with label $\sigma \in \Sigma$ the associated node in ρ with label $s \in S$ has children $s_1, ..., s_{|\sigma|}$ such that $\left(s_1, ..., s_{|\sigma|}\right) \in \delta(s)(\sigma)$. A run is accepted if for every branch of the tree, the maximum priority that occurs infinitely is even. A tree $\tau \in \mathrm{Tree}_\Sigma$ is accepted by A if there is an accepting run of A on τ. The accepted language of A is the set of accepted trees.

2.3 Fixed Points

Crucial for the next section, Section 2.4 about modal μ-calculus, is reasoning about *fixed points* of *monotone* functions. We briefly recall the important definitions and theorems.

Definition 2.3: A *complete lattice* is a partially ordered set $\langle L, \leq \rangle$ such that every subset $M \subseteq L$ has a least upper bound $\bigvee M$ and greatest lower bound $\bigwedge M$. Specifically, the whole set L has a least and greatest element, which we denote $\bigwedge L = \bot$ and $\bigvee L = \top$, respectively.

In this report we usually deal with the powerset of some set where subsets are ordered by inclusion. For a set S, $\langle \mathcal{P}(S), \subseteq \rangle$ is a complete lattice. For $U \subseteq \mathcal{P}(S)$, $\bigvee U := \bigcup U$, and $\bigwedge U := \bigcap U$. The least and greatest elements are $\emptyset$ and S, respectively. The following is known as the Knaster-Tarski Fixed Point Theorem [1:Theorem 1.2.8]:

Theorem 2.4: Let $\langle L, \leq \rangle$ a complete lattice and $f : L \to L$ monotone ($f(x) \leq f(y)$ when $x \leq y$). Then, the set of fixed points $\{x \in L | f(x) = x\}$, is a complete lattice. Particularly, the function has a *least fixed point* (lfp), denoted $\mathrm{lfp}(f)$, and a *greatest fixed point* (gfp), denoted $\mathrm{gfp}(f)$.

There is a useful way of constructing these least and greatest fixed points. This is done by repeated function application on $\bot$ for the least fixed point, and $\top$ for the greatest fixed point. Concretely, we define for a monotone $f : L \to L$, for α an ordinal, and β a limit ordinal:

$$\begin{aligned} f^0 &:= \bot \\ f^{\alpha+1} &:= (f^\alpha) \\ f^\beta &:= \bigvee \{f^\alpha \mid \alpha < \beta\} \end{aligned} \tag{1}$$

This constructs an increasing chain

$$\bot = f^0 \leq f^1 \leq f^2 \leq ... \tag{2}$$

which eventually stabilizes, giving the least fixed point, as stated by the following theorem:

Theorem 2.5: There exists an ordinal κ, such that $f^\kappa = f^{\kappa+1}$, which implies that f^κ is a fixed point of f. Furthermore, $f^\kappa = \mathrm{lfp}(f)$ is the least fixed point of f. The dual process, beginning from $\top$ and moving downward, constructs the greatest fixed point of f.

2.4 Modal μ-Calculus

The modal μ-calculus is a powerful logic, used to verify properties of transition systems [1, 3]. We use it in Section 3.2 to select the right accepting trees for the coalgebraic system. In this section we give a concrete definition of modal μ-calculus formulas and provide intuition on how to use the modal μ-calculus to verify certain properties. We verify these properties over *transition systems*, which we define first:

Definition 2.6: A transition system (TS) is a tuple $T = \langle S, \delta, Prop, \lambda \rangle$ where S is the set of states, $\delta \subseteq S \times S$ the transition relation (we sometimes write $s \to s'$ if $(s, s') \in R$), $Prop$ the set of atomic propositions, and $\lambda : Prop \to \mathcal{P}(S)$ interprets the atomic propositions.

A TS can be seen as a directed graph where the vertices are labeled by atomic propositions $\mathcal{P}(Prop)$. Note that usually the modal μ-calculus is defined on *labeled* transition systems, but to simplify things slightly, and because we only need unlabeled transition systems in the rest of the report we stick to transition systems.

Next we define the syntax of the modal μ-calculus:

Definition 2.7: A modal μ-calculus formula is defined by the grammar:

$$\varphi \coloneqq P \mid Z \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Box\varphi \mid \Diamond\varphi \mid \mu Z.\varphi \mid \nu Z.\varphi \tag{3}$$

where $P \in Prop$ is an atomic proposition and $Z \in Var$ a *fixed point variable.*

Note that we could define the modal μ-calculus without the $\vee$, $\Diamond$, and ν operators, and define these instead in terms of the other operators, but we include them in the definition for legibility.

Definition 2.8: For a modal μ-calculus formula φ, a transition system $T = \langle S, \delta, Prop, \lambda\rangle$ and an assignment $V : Var \to \mathcal{P}(S)$ we define the semantics $\|\varphi\|_V^T \subseteq S$ of the formula φ as follows:

$$\begin{aligned}
\|P\|_V^T &\coloneqq \lambda(P) \\
\|\neg P\|_V^T &\coloneqq S \setminus \lambda(P) \\
\|Z\|_V^T &\coloneqq V(Z) \\
\|\ \varphi_1 \wedge \varphi_2\ \|_V^T &\coloneqq \|\varphi_1\|_V^T \cap \|\varphi_2\|_V^T \\
\|\ \varphi_1 \vee \varphi_2\ \|_V^T &\coloneqq \|\varphi_1\|_V^T \cup \|\varphi_2\|_V^T \\
\|\ \Box\varphi\ \|_V^T &\coloneqq \{s \mid \forall t \in S.\ \text{if } s \to t \text{ then } t \in \|\varphi\|_V^T\} \\
\|\ \Diamond\varphi\ \|_V^T &\coloneqq \{s \mid \exists t \in S. s \to t \text{ and } t \in \|\varphi\|_V^T\} \\
\|\ \mu Z.\varphi\ \|_V^T &\coloneqq \mathrm{lfp}\big(\lambda U.\ \|\varphi\|_{V[Z\mapsto U]}^T\big) \\
\|\ \nu Z.\varphi\ \|_V^T &\coloneqq \mathrm{gfp}\big(\lambda U.\ \|\varphi\|_{V[Z\mapsto U]}^T\big)
\end{aligned} \tag{4}$$

where $V[Z \mapsto U]$ is the valuation V except that Z maps to U.

We write $s \vDash^T \varphi$ if $s \in \|\varphi\|_V^T$ for an empty valuation V, or just $s \vDash \varphi$ if T is clear.

Let us briefly look at some intuition behind these definitions. We have $s \vDash^T p$ if in T at state s the propositional variable p holds. Conversely,$s \vDash^T \neg p$ holds if p does not hold in s. The $\Diamond$ and $\Box$ operators look at states reachable from s. For example, $s \vDash^T \Diamond p$ is true if there is some state s' such that $s \to s'$ and $s' \vDash^T p$. Analagously, $s \vDash^T \Box p$ is true if p is true in all succesor states from s. Less intuitive are the μ and ν operators. Concretely, they identify least and greatest fixed points on functions from states to states. More intuitively, they can be used to define looping properties on transition systems, where μ can be used for finite looping, and ν for infinite looping. This will hopefully become more clear when looking at some examples:

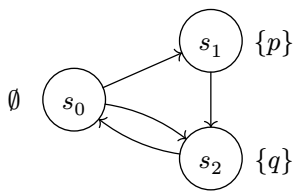

Figure 2: Example of a TS. The sets next to the states denote the atomic propositions that hold in that state.

Consider the transition system given in Figure 2. We have $s_0 \vDash \Diamond p$, because there is a transition from s_0 to a state where p holds, namely $s_0 \to s_1$, because $s_1 \vDash p$. We, however, do not have $s_0 \vDash \Box p$, because $s_0 \to s_2$ and $s_2 \nvDash p$.

To observe that μ is associated with finite looping, we look at the fact that $s_0 \vDash \mu Z.q \vee \Box Z$. This means that all finite paths from s_0 either eventually reach a state with no outgoing transitions, or reach a state where q is true. We can see in Figure 2 that from s_0 every path reaches a state where q is true in finitely many steps. To

more formally show that this holds, we make use of the method of constructing least and greatest fixed points in Theorem 2.5. The function we are calculating the lfp for is $f := \lambda U.\ \|q\| \cup \|\Box U\|$. The first iteration yields $f^1 = f(\emptyset) = \{s_2\}$, because $s_2 \vDash q$. Continuing, $f^2 = \{s_1, s_2\}$ and $f^3 = \{s_0, s_1, s_2\} = f^4$. So the lfp is the entire set of states S, and thus $s_0 \vDash \mu Z.q \vee \Box Z$.

Next we look at ν, which can be used for infinite looping. We show that $s_0 \vDash \nu Z.\Diamond Z$. This intuitively means that there exists an infinite path from s_0. Indeed, we observe there are multiple infinite paths starting from s_0. We confirm by computing the gfp: $f^1 = f(S) = \Diamond S = S$. Dually, observe that the lfp of this formula is $f^1(\emptyset) = \emptyset$. So we do not have $s_0 \vDash \mu Z.\Diamond Z$. This confirms the intuition that μ is for finite looping: there has to be some end point of the loop.

2.4.1 System of Equations

Next we introduce systems of equations with alternating fixed points. We demonstrate the system using only two equations for brevity and because they are the only ones needed in the rest of the report. For more detail into this specific topic see [1, 9].

Definition 2.9: Let L_1, L_2 be complete lattices. An *equational system* is a system of two equations

$$u_1 \underset{\eta_1}{=} f_1(u_1, u_2) \qquad u_2 \underset{\eta_2}{=} f_2(u_1, u_2) \tag{5}$$

where u_1, u_2 are variables, $\eta_1, \eta_2 \in \{\mu, \nu\}$, and $f_i : L_1 \times L_2 \to L_i$ are monotone functions. The solution to the system is defined by the following set of steps:

The intermediate solution $l_1^{(1)} := \eta_1 u_1.f_1(u_1, u_2)$, where we take the lfp if $\eta_1 = \mu$ and gfp if $\eta_1 = \nu$. Note that $l_1^{(1)} : L_2 \to L_1$.

The solution to the second equation is then given by $l^{\text{sol}} := \eta_2 u_2.f_2\left(l_1^{(1)}(u_2), u_2\right)$, where again we take the lfp if $\eta_2 = \mu$, and gfp if $\eta_2 = \nu$. The solution to the first equation is then $l_1^{\text{sol}} = l_1^{(1)}(l_2^{\text{sol}})$.

2.5 Parity Games

Next we introduce parity games and show how they provide both an intuitive and formal semantics for modal μ-calculus formulas. We use these semantics to prove the coincidence results in Section 4.

A parity game is a two player game between V (verifier) and R (refuter), who want to verify, respectively refute, a statement. In our case, this statement is $s \vDash^T \varphi$, i.e. that a modal μ-calculus formula holds in a state s in LTS T. So V wants to show $s \vDash^T \varphi$ and R wants to show $s \nvDash^T \varphi$. The game consists of states and transitions between these states. Every state 'belongs' to either V or R, which determines which player picks the next transition and thus the next state. A play of the game is then a (possibly infinite) sequence of states, and is won by either V or R. Concretely we define:

Definition 2.10: A *parity game* is a tuple $((S_V, S_R), E, \Omega)$, where $S = S_V \sqcup S_R$ is the set of states, partitioned between player V (S_V) and player R (S_R) who choose transitions from their repsective states, $E \subseteq S \times S$ are transitions between the states, and $\Omega : S \to \omega$ is the parity function, which determines the winner for infinite plays.

A play of the game is a (possibly infinite) sequence of states $s_1, s_2, \ldots$ such that $(s_i, s_{i+1}) \in E$. A finite play $s_1, s_2, \ldots, s_n$ is won by V if $s_n \in S_R$ and there is no s_{n+1} such that $(s_n, s_{n+1}) \in E$, i.e. player R has no moves. Analagously, R wins if player V gets stuck. An infinite play $\pi = s_1, s_2, \ldots$ is won by V if $\max\{\Omega(s) \mid s \text{ occurs infinitely often in } \pi\}$ is even, and won by R if it is odd.

Next, we introduce the parity game for the modal μ-calculus. Consider the formula $\varphi = \varphi_1 \vee \varphi_2$. V wants to verify $s \vDash^T \varphi$, and to do so it suffices to show for either φ_i that $s \vDash^T \varphi_i$. Analagously for the formula $\varphi = \varphi_1 \wedge \varphi_2$, R can 'pick' the φ_i such that $s \nvDash^T \varphi_i$, because if either φ_1 or φ_2 does not hold, φ does not hold. This same duality is seen in $\Diamond\varphi$ and $\Box\varphi$ where for $\Diamond$ V can show there is a transition for which φ holds, and for $\Box\varphi$, R can pick a transition such that φ does not hold. This way the game arises between V and R to determine whether $s \vDash^T \varphi_i$:

Definition 2.11: For a transition system $T = (S, \delta, Prop, \lambda)$ and a modal μ-calculus formula φ, we define the game $\mathcal{G}(\varphi, T) = ((S_V, S_R), E, \Omega)$ where:

- The states of the game $S_V \sqcup S_R = \{\varphi' \mid \varphi' \text{ is a subformula of } \varphi\} \times S$ are pairs of a subformula of φ and a state in the TS. The subformula determines to which player the state belongs to. For a subformula ψ and a state s of the TS:
  - $(\psi, s) \in S_V$ if
    - $\psi = \psi_1 \vee \psi_2$
    - $\psi = \Diamond\psi'$
    - $\psi = \eta Z.\psi'$ for $\eta \in \{\mu, \nu\}$
    - $\psi = Z$ for Z a fixed point variable
    - $\psi = p$ for p a propositional variable with $s \notin \lambda(p)$.
    - $\psi = \neg p$ for p a propositional variable with $s \in \lambda(p)$.
  - $(\psi, s) \in S_R$, if
    - $\psi = \psi_1 \wedge \psi_2$
    - $\psi = \Box\psi'$
    - $\psi = p$ for p a propositional variable with $s \in \lambda(p)$.
    - $\psi = \neg p$ for p a propositional variable with $s \notin \lambda(p)$.
- Edges E:
  - $(\psi_1 \vee \psi_2, s) \to (\psi_1, s)$ and $(\psi_1 \vee \psi_2, s) \to (\psi_2, s)$
  - $(\psi_1 \wedge \psi_2, s) \to (\psi_1, s)$ and $(\psi_1 \wedge \psi_2, s) \to (\psi_2, s)$
  - $(\Diamond\psi, s) \to (\psi, s')$ for any s' such that $s \to s'$ in T.
  - $(\Box\psi, s) \to (\psi, s')$ for any s' such that $s \to s'$ in T.
  - $(\eta Z.\psi, s) \to (\psi, s)$ and $(Z, s) \to (\psi, s)$ for $\eta \in \{\mu, \nu\}$
- The priority function Ω depends on the *alternation depth* $\alpha(\psi)$ of the subformula ψ, which is defined as follows:
  - $\alpha(p) = \alpha(\neg p) = 0$ for p a propositional variable
  - $\alpha(\psi_1 \wedge \psi_2) = \alpha(\psi_1 \vee \psi_2) = \max\{\alpha(\psi_1), \alpha(\psi_2)\}$
  - $\alpha(\Diamond\psi) = \alpha(\Box\psi) = \alpha(\psi)$
  - $\alpha(\mu Z.\psi) = \max(\{1, \alpha(\psi)\} \cup$
    $\{\alpha(\nu Z'.\psi') + 1 \mid \nu Z'.\psi' \text{ is a subformula of } \psi \text{ and } Z \text{ occurs free in } \psi'\})$
  - $\alpha(\nu Z.\psi) = \max(\{1, \alpha(\psi)\} \cup$
    $\{\alpha(\mu Z'.\psi') + 1 \mid \mu Z'.\psi' \text{ is a subformula of } \psi \text{ and } Z \text{ occurs free in } \psi'\})$

  Intuitively, the alternation depth of a formula is the maximum number of alternating μ/ν operators, where we only count those alternations where the free variable actually occurs freely in the subformula, meaning the fixed point operators are actually interdependent. Ω is then:
  - $\Omega((\mu Z.\psi, s)) =$ the smallest odd number greater or equal than $\alpha(\psi) - 1$
  - $\Omega((\nu Z.\psi, s)) =$ the smallest even number greater or equal than $\alpha(\psi) - 1$
  - $\Omega((\psi, s)) = 0$ iff ψ is not a μ or ν formula.

Whereas the intuition for operators like $\vee, \wedge, \Box, \Diamond$ is quite straightforward, for the μ/ν operators it is less so. Briefly put, it follows from what was explained in Section 2.4 that μ incites finite looping, and ν infinite looping. It can be seen from the definition for Ω using the alternation depth, that outer μ/ν operators have higher priority than inner ones, and μ is always even and ν odd. Thus the highest priority occuring infinitely often in an infinite play indicates the outermost fixed point operator that is visited infinitely often. Thus, if this is even, we have an infinite loop through a ν operator, which satisfies the formula. For a μ operator, however, an infinite loop is undesired, and thus if the outermost fixed point operator which is visited infinitely often is μ, R has refuted the formula.

Now, to use this game to give alternative semantics for the modal μ-calculus we need that $s \vDash^T \varphi$ if and only if V can verify this in the game $\mathcal{G}(\varphi, T)$ by winning the game, and R can not win. We say that V has a winning strategy: V can always play (i.e. take the right transition if it is their turn) such that regardless of how R plays, V wins the play. We then have the crucial theorem (for the proof see [3:Theorem 10.18]) for our derivation of the concidence results in Section 4:

Theorem 2.12: For a transition system $T = \langle S, \delta, Prop, \lambda \rangle$, a state $s \in S$, and a modal μ-calculus formula φ, we have:

$$s \vDash^T \varphi \Leftrightarrow \text{V has a winning strategy in } \mathcal{G}(\varphi, T) \text{ starting in state } (\varphi, s) \tag{6}$$

3 Coalgebraic Representation of Büchi Automata

3.1 Finite Behavior of Nondeterministic Automata

In this section we present a coalgebraic representation of nondeterministic systems. The next section for Büchi automata builds upon this construction.

3.1.1 Deterministic Automata

First we consider a deterministic finite automaton, $\langle S, \Sigma, \delta, o \rangle$ with S the states, Σ the alphabet, $\delta : S \times \Sigma \to S$ the transition function, and $o : S \to 2$ with $2 = \{0, 1\}$, the output function determining if a state is final. We do not consider an initial state here because we just want to obtain the accepted words for each state. Such an automaton can be represented by a coalgebra $c : S \to 2 \times S^\Sigma$ for the functor $FS = 2 \times S^\Sigma$. This is a very useful construction because a final coalgebra for this functor is carried by 2^{Σ^*}, and the unique coalgebra homomorphism beh to this final coalgebra captures exactly the language accepted by a state [8]. This is shown in the commuting diagram:

$$\begin{tikzcd}
2 \times S^\Sigma \arrow[r, dashed, "\mathrm{id} \times \mathsf{beh}^\Sigma"] & 2 \times \left(2^{\Sigma^*}\right)^\Sigma \\
S \arrow[u, "\langle o, \delta \rangle"] \arrow[r, dashed, "\mathsf{beh}"'] & 2^{\Sigma^*} \arrow[u, "\langle e, d \rangle"']
\end{tikzcd} \tag{7}$$

Here $e : 2^{\Sigma^*} \to 2$ is given by $e(L) = L(\varepsilon)$, i.e., $e(L) = 1$ iff L contains the empty word. And $d : 2^{\Sigma^*} \to \left(2^{\Sigma^*}\right)^\Sigma$ is the language derivative, given by $d(L)(\sigma) = L_\sigma$ where $L_\sigma(w) = L(\sigma w)$, so $w \in d(L)(\sigma) = L_\sigma$ iff $\sigma w \in L$.

Working out the paths through the diagram we obtain that

- $\mathsf{beh}(s)(\varepsilon) = o(s)$, and
- $\mathsf{beh}(s)(\sigma w) = \mathsf{beh}(\delta(s)(\sigma))(w)$,

for $s \in S$, $\sigma \in \Sigma$, $w \in \Sigma^*$. So $\mathsf{beh}(s)$ contains the empty word iff s is a final state, and accepts σw iff $\delta(s)(\sigma)$ accepts w. Which is precisely the language accepted by state x in the deterministic finite automaton!

3.1.2 Nondeterministic Automata

Unfortunately, extending this approach to nondeterministic automata is not possible, as we will illustrate by the following automaton, which we will use as a running example:

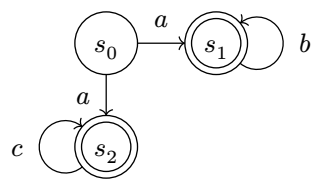

Figure 3: Example of a nondeterministic automaton.

The automaton given in Figure 3 is nondeterministic because in s_0 there are two transitions for a. Intuitively, the finite words accepted by the automaton from state s_0 should be

$$\mathsf{beh}(s_0) = \{a, ab, abb, ...\} \cup \{a, ac, acc, ...\}. \tag{8}$$

This transitions system might be modeled by a coalgebra $c : S \to 2 \times \mathcal{P}(\Sigma \times S)$, i.e., for every state whether it is final, and a set of pairs $(\sigma, s) \in \Sigma \times S$ denoting a transition by taking letter σ and transitioning to state s. The problem is that this functor $FS = 2 \times \mathcal{P}(\Sigma \times S)$ does not have a final coalgebra, as Lambek's lemma implies that such a final coalgebra structure $z : Z \to 2 \times \mathcal{P}(\Sigma \times Z)$ for some carrier Z, would have to be an

isomorphism [2]. But an isomorphism $Z \cong 2 \times \mathcal{P}(\Sigma \times Z)$ would imply a bijection between Z and $\mathcal{P}(Z)$, which cannot exist.

The solution, as given by Hasuo et al. [4], is to work in the Kleisli category for the monad $\mathcal{P}$. Recall that a map $f : X \to Y$ in the Kleisli category is a map $f : X \to \mathcal{P}(Y)$ in **Sets**. Briefly put, this will solve our problem because we can have a final coalgebra $z : Z \to FZ$ that is a map $z : Z \to \mathcal{P}(FZ)$ in **Sets**. Next, we will review the definition of the Kleisli category and define the appropriate functor, enabling us to construct the desired final coalgebra that characterizes the accepting finite words.

The powerset monad $\mathcal{P}$ is defined by the unit $\eta_X : X \to \mathcal{P}(X)$ which sends an element of X to the singleton set, $\eta_X(x) = \{x\}$ for $x \in X$, and the multiplication $\mu_X : \mathcal{P}(\mathcal{P}(X)) \to \mathcal{P}(X)$ which takes the union of the sets, i.e. $\mu_X(A) = \bigcup_{a \in A} a$. For a function $f : X \to Y$ we get $\mathcal{P}(f) : \mathcal{P}(X) \to \mathcal{P}(Y)$ by $\mathcal{P}(f)(A) = \{f(a) \mid a \in A\}$. The Kleisli category for this monad is defined as follows:

- **objects**: the same as for **Sets**, sets
- **morphisms**: a morphism f from X to Y in $\mathcal{K}\ell(\mathcal{P})$ is a map $f : X \to \mathcal{P}(Y)$ in **Sets**.

  The identity morphism $\mathrm{id}_X : X \to X$ is defined by $\mathrm{id}_{X(x)} = \{x\}$, which is indeed a mapping from X to $\mathcal{P}(X)$ in **Sets**. Composition $(g \circ f)$ in $\mathcal{K}\ell(\mathcal{P})$ for morphisms $f : X \to Y$ and $g : Y \to Z$ in $\mathcal{K}\ell(\mathcal{P})$ (so $f : X \to \mathcal{P}(Y)$ and $g : Y \to \mathcal{P}(Z)$ in **Sets**) is defined as $(\mu_Z \circ \mathcal{P}(g) \circ f)$ in **Sets**. Again we see that that this definition is of the right type as $(\mu_Z \circ \mathcal{P}(g) \circ f) : X \to \mathcal{P}(Z)$ in **Sets**, so $(g \circ f) : X \to Z$ in $\mathcal{K}\ell(\mathcal{P})$.

Next, we construct our functor in $\mathcal{K}\ell(\mathcal{P})$, which we call the lifting of F in $\mathcal{K}\ell(\mathcal{P})$, and denote $\overline{F}$. The key here is that because we are working in the Kleisli category, if we use the functor $\overline{F}S = \Sigma \times S$, the coalgebra map $c : S \to \Sigma \times S$, will be a map $c : S \to \mathcal{P}(\Sigma \times S)$ in **Sets**, which models nondeterministic transitions. In the previous section we used the functor $FS \to 2 \times S^{\Sigma}$ where $o : S \to 2$ denoted whether the state was final. Combining this with the functor $\overline{F}S = \Sigma \times S$ in the Kleisli category we would get the functor $\overline{F}S = 2 \times \Sigma \times S$ and the coalgebra $c : S \to 2 \times \Sigma \times S$ which is $c : S \to \mathcal{P}(2 \times \Sigma \times S)$ in **Sets**, which would mean that every transition can be final or not, which is not what we want. For this reason we use the functor $\overline{F}S = 1 + \Sigma \times S$ such that the coalgebra $c : S \to 1 + \Sigma \times S$ is the map $c : S \to \mathcal{P}(1 + \Sigma \times S)$ where $s \in S$ is final iff $* \in c(s)$ (note that we use $1 = \{*\}$).

This works easily on objects, $\overline{F}X = FX$, because in the Kleisli category, the objects are the same. But for morphisms we have to do a little bit more work. Observe that because a map $f : X \to Y$ in $\mathcal{K}\ell(\mathcal{P})$ is a map $f : X \to \mathcal{P}(X)$ in **Sets**, applying the functor F on the map itself would yield $Ff : FX \to F\mathcal{P}(Y)$. So what we need is a natural transformation $\lambda : F\mathcal{P} \Rightarrow \mathcal{P}F$ (this is a distributive law, for more details see for example [4]) such that $1 + \Sigma \times (\mathcal{P}(S)) \xrightarrow{\lambda} \mathcal{P}(1 + \Sigma \times S)$. We define this as $* \mapsto \{*\}$, and $(\sigma, U) = \{(\sigma, s) | s \in U\}$ for $\sigma \in \Sigma$ and $U \subseteq S$. We see that this definition makes sense if we observe that if taking transition σ from state s takes you to $\{x, y, z\}$, i.e. $(\sigma, \{x, y, z\}) \in c(s)$, or $\{x, y, z\} \in \delta(s)(\sigma)$, you can also see this as a set of transitions $\{(\sigma, x), (\sigma, y), (\sigma, z)\}$.

Finally, the main theorem from [4] (Theorem 3.3), and the last ingredient to make the construction work, is that the initial algebra for the functor F in **Sets** gives us the final coalgebra for the lifted functor $\overline{F}$ in $\mathcal{K}\ell(\mathcal{P})$. Specifically, for this functor $FS = 1 + \Sigma \times S$ and its lifting as described above, the initial F-algebra $\alpha : FA \to A$ in **Sets** yields a final $\overline{F}$-coalgebra in $\mathcal{K}\ell(\mathcal{P})$ by:

$$(\eta_{FA}\alpha)^{-1} = \eta_{FA}\alpha^{-1} : A \to \overline{F}A \ in \ \mathcal{K}\ell(\mathcal{P}) \tag{9}$$

In fact, this result holds more generally: for the lifting monad $\mathcal{L}$, the subdistribution monad $\mathcal{D}$, and any shapely functor F, see [4] for more details.

The initial F-algebra for our functor $FS = 1 + \Sigma \times S$ in **Sets** is $[\mathsf{nil}, \mathsf{cons}] : 1 + \Sigma \times \Sigma^* \to \Sigma^*$. So we get the commuting diagram

$$
\begin{tikzcd}
1+\Sigma\times S \arrow[r, dashed, "1+\Sigma\times\mathsf{beh}"] & 1+\Sigma\times\Sigma^* \\
S \arrow[u, "c"] \arrow[r, dashed, "\mathsf{beh}"'] \arrow[ur, phantom, "\cong"] & \Sigma^* \arrow[u, "\eta_{1+\Sigma\times\Sigma^*}\circ[\mathsf{nil},\mathsf{cons}]^{-1}"']
\end{tikzcd}
\quad in\ \mathcal{K}\ell(\mathcal{P}). \tag{10}
$$

Following the paths within the diagram we obtain that

$$
\begin{aligned}
\varepsilon \in \mathsf{beh}(s) &\iff * \in c(s) \iff \text{state } s \text{ is accepting} \\
\sigma w \in \mathsf{beh}(s) &\iff \exists t.\big(s \xrightarrow{\sigma} t \wedge w \in \mathsf{beh}(t)\big).
\end{aligned} \tag{11}
$$

Explained in words, a state accepts the empty word iff the state is accepting, and it accepts σw for $\sigma \in \Sigma$ and $w \in \Sigma^*$ iff it can transition with σ to a state which accepts w. Which is exactly the desired words!

3.1.3 Possibly Infinite Behavior

As a step towards infinite words in Büchi automata let us consider infinite words in Figure 3. We can slightly alter our previous construction to additionally obtain infinite words through this system [4, 5]. Concretely, the infinite words for the system in Figure 3 for x_0 are ab^ω and ac^ω.

The intuition for this new construction is as follows. In the previous section we constructed the final coalgebra for the lifted functor $\overline{F}$ using the initial F-algebra in **Sets**. In the example of the LTS with termination the initial algebra was carried by Σ^*. The final coalgebra in **Sets** for F is carried by Σ^∞ (where the ∞ operators means some finite number of times or indefinitely so $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$) the set of finite and infinite words. So if we use this final coalgebra instead of the initial algebra, do we obtain both the finite and infinite words?

Consider again the monad $\mathcal{P}$, our functor F (this too holds more generally, see [4, 5]), and its lifting in the Kleisli category $\overline{F}$. For a final coalgebra $\xi : Z \to FZ$, the coalgebra

$$
\eta_{FZ} \circ \xi : Z \to \overline{F}Z \ in\ \mathcal{K}\ell(\mathcal{P}) \tag{12}
$$

is *weakly final*. That means, for any coalgebra $c : S \to \overline{F}S$, there is a morphism $\mathsf{beh} : S \to Z$ in $\mathcal{K}\ell(\mathcal{P})$ such that the following diagram commutes

$$
\begin{tikzcd}
\overline{F}S \arrow[r, squiggly, "\overline{F}(\mathsf{beh})"] & \overline{F}Z \\
S \arrow[u, "c"] \arrow[r, squiggly, "\mathsf{beh}"'] \arrow[ur, phantom, "\cong"] & Z \arrow[u, "\eta_{FZ}\circ\xi"']
\end{tikzcd}
\quad in\ \mathcal{K}\ell(\mathcal{P}), \tag{13}
$$

but this morphism is not necessarily unique. However, there is a canonical choice beh^∞ among these morphisms, namely the one which is maximal with respect to inclusion. We call this function $\mathsf{beh}^\infty : S \to \mathcal{P}(Z)$ (in **Sets**) the *possibly-infinite* behavior for c.

Indeed, if we consider our running example Figure 3 with termination, $\xi : \Sigma^\infty \to 1 + \Sigma \times \Sigma^\infty$ is the final F-coalgebra, defined by $\xi(\varepsilon) = * \in 1$ and $\xi(\sigma w) = (\sigma, w)$. Instantiating the diagram in Equation 13, we obtain

$$
\begin{aligned}
\varepsilon \in \mathsf{beh}^\infty(s) &\iff * \in c(s) \iff \text{state } s \text{ is accepting} \\
\sigma w \in \mathsf{beh}^\infty(s) &\iff \exists t.\big(s \xrightarrow{\sigma} t \wedge w \in \mathsf{beh}^\infty(t)\big).
\end{aligned} \tag{14}
$$

Which is the same as in Equation 11. However, because the domain is Σ^∞, we obtain different words when we take the maximal function satisfying these equations. Namely the infinite words, in addition to the finite ones! For the system in Figure 3 we get the same words as before, but additionally $\{ab^\infty, ac^\infty\} \subseteq \mathsf{beh}^\infty_c(s_0)$. Interestingly, taking the minimum morphism we again obtain just the finite words [4, 5].

3.2 Coalgebraic Representation of Büchi Automata

We can apply the previous framework for possibly infinite words to our initial example for a Büchi automaton, in Figure 1. This would yield all infinite words through the automaton, so also for example $\texttt{request} \cdot \texttt{process}^{\omega}$. Meaning, it only takes into account accepting states for ending finite words. How do we eliminate those words that process indefinitely? That is, only accept those words under the Büchi acceptance criterion of passing through an accepting state infinitely many times.

A way of solving this is given by [9]. In short, the main idea of this paper is to divide the states into accepting and non-accepting states and applying the previous construction using the final F-coalgebra in **Sets** to obtain two seperate commuting diagrams. Then, by using greatest and least fixed points we can precisely pick exactly the accepting words for the Büchi automaton.

We first give the commuting diagrams which govern the behavior mappings. We now consider the functor $FS = \Sigma \times S$. Because, the final coalgebra for this functor is (Σ^{∞}, d) where d is defined by $d(\sigma \cdot w) = (\sigma, w)$, we consider only the infinite words. The lifting $\overline{F}$ is effectively the same, just without a case for $* \in 1$. We now consider the state space as a disjoint union $S = S_1 \cup S_2$ of non-accepting and accepting states, respectively. This gives rise to two separate coalgebras $c_i : S_i \to \overline{F}X$, defined by the restriction $c \circ \kappa_i : S_i \to \overline{F}X$ along the coprojection $\kappa_i : S_i \hookrightarrow S$ for $i \in \{1, 2\}$. We then get the two commuting diagrams:

$$\begin{tikzcd} \Sigma \times [\mathsf{beh}_1, \mathsf{beh}_2] & \\ \Sigma \times S \arrow[r, squiggly] & \Sigma \times \Sigma^{\omega} \\ S_1 \arrow[u, "c_1"] \arrow[r, squiggly, "\mathsf{beh}_1"'] \arrow[ur, phantom, "\underset{\mu}{=}"] & \Sigma^{\omega} \arrow[u, "\eta_{\Sigma^{\omega}} \circ d"'] \arrow[ul, phantom, "\cong"] \end{tikzcd} \qquad \begin{tikzcd} \Sigma \times [\mathsf{beh}_1, \mathsf{beh}_2] & \\ \Sigma \times S \arrow[r, squiggly] & \Sigma \times \Sigma^{\omega} \\ S_2 \arrow[u, "c_2"] \arrow[r, squiggly, "\mathsf{beh}_2"'] \arrow[ur, phantom, "\underset{\nu}{=}"] & \Sigma^{\omega} \arrow[u, "\eta_{\Sigma^{\omega}} d"'] \arrow[ul, phantom, "\cong"] \end{tikzcd} \; in\ \mathcal{K}\ell(\mathcal{P}). \tag{15}$$

Where $\underset{\mu}{=}$ and $\underset{\nu}{=}$ mean that we take the least behavior mapping in the left diagram to obtain beh_1, and the greatest behavior mapping in the right diagram to obtain beh_2. More concretely, $\mathsf{beh}_1 : S_1 \to \mathcal{P}(\Sigma^{\omega})$ and $\mathsf{beh}_2 : S_2 \to \mathcal{P}(\Sigma^{\omega})$, are the solutions to the following system of equations:

$$\begin{aligned} u_1 &\underset{\mu}{=} (\eta_{\Sigma^{\omega}} \circ d)^{-1} \odot \overline{F}[u_1, u_2] \odot c_1 \\ u_2 &\underset{\nu}{=} (\eta_{\Sigma^{\omega}} \circ d)^{-1} \odot \overline{F}[u_1, u_2] \odot c_2 \end{aligned} \tag{16}$$

At this point we note that this construction works for the more general parity tree automata from Section 2.2. A parity tree automata can be modeled by the lifted functor of the functor $FS = \bigsqcup_{\sigma \in \Sigma} S^{|\sigma|}$ (where $\sqcup$ is the coproduct). The final coalgebra of F in **Sets** is $d : \mathrm{Tree}_{\Sigma} \to \bigsqcup_{\sigma \in \Sigma} \mathrm{Tree}_{\Sigma}^{|\sigma|}$ where $d\big(\big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big)\big) = \big(\tau_1, ..., \tau_{|\sigma|}\big) \in S^{|\sigma|}$. We model the different parities by splitting the states into $S = S_1 \cup ... \cup S_n$ where for every S_i we have $s \in S_i \to \Omega(s) = i$ and we have a seperate commuting diagram like in Equation 15 for every parity, where the μ and ν alternate. We then get the following equations for a parity tree automaton with n parities:

$$\begin{aligned} u_1 &\underset{\mu}{=} \big(\eta_{\mathrm{Tree}_{\Sigma}} \circ d\big)^{-1} \odot \overline{F}[u_1, ..., u_n] \odot c_1 \\ u_2 &\underset{\nu}{=} \big(\eta_{\mathrm{Tree}_{\Sigma}} \circ d\big)^{-1} \odot \overline{F}[u_1, ..., u_n] \odot c_2 \\ &\vdots \\ u_n &\underset{\eta_n}{=} \big(\eta_{\mathrm{Tree}_{\Sigma}} \circ d\big)^{-1} \odot \overline{F}[u_1, ..., u_n] \odot c_n \end{aligned} \tag{17}$$

where $\eta_i = \mu$ if i is odd and $\eta_i = \nu$ if i is even. We confirm here that the Büchi case is a specific instance of the parity acceptence criterion by letting $n = 2$ and just having one μ and one ν equation. Before continuing, we rewrite the equations to be more clear and usable:

Lemma 3.1: The equations in Equation 17 coincide with:

$$u_1 \overset{\mu}{=} \Diamond_\delta([u_1, ..., u_n]) \restriction S_1, \quad ..., \quad u_n \overset{\eta_n}{=} \Diamond_\delta([u_1, ..., u_n]) \restriction S_n \tag{18}$$

where $\eta_i = \mu$ if i is odd and $\eta_i = \nu$ if i is even and $\Diamond_\delta : (\mathcal{P}(\mathrm{Tree}_\Sigma))^S \to (\mathcal{P}(\mathrm{Tree}_\Sigma))^S$ is given by

$$\Diamond_\delta(\mathsf{beh})(s) = \left\{ \left(\sigma, \left(\tau_1, ..., \tau_{|\sigma|}\right)\right) \mid \sigma \in \Sigma, \left(s'_1, ..., s'_{|\sigma|}\right) \in \delta(s)(\sigma), \tau_i \in \mathsf{beh}(s'_i) \right\}. \tag{19}$$

The proof can be found in Appendix A.1.

By taking exactly those behavior mappings which are the solution to this system of equation, we take exactly those words that the parity tree automaton accepts. This is given by the following lemma, which is Lemma 4.5 in [9]:

Lemma 3.2: Let $A = (S, \Sigma, \delta, s_0, \Sigma)$ be a parity tree automaton, where we let $S = S_1 \cup ... \cup S_n$ the disjoint union of states with parity $1, ..., n$, respectively. Let l_i^{sol} be the solutions to the following equational system, where the variables $u_1, ..., u_n$ range over $(\mathcal{P}(\mathrm{Tree}_\Sigma))^{S_i}$

$$u_1 \overset{\mu}{=} \Diamond_\delta([u_1, ..., u_n]) \restriction S_1, \quad ..., \quad u_n \overset{\eta_n}{=} \Diamond_\delta([u_1, ..., u_n]) \restriction S_n \tag{20}$$

where $\eta_i = \mu$ if i is odd and $\eta_i = \nu$ if i is even and $\Diamond_\delta : (\mathcal{P}(\mathrm{Tree}_\Sigma))^S \to (\mathcal{P}(\mathrm{Tree}_\Sigma))^S$ is as in Equation 19. Then the solutions $l_i^{\mathrm{sol}} : S_i \to \mathcal{P}(\mathrm{Tree}_\Sigma)$ map each state in S_i to the accepted language from that state, that is, $l_i^{\mathrm{sol}}(s) = L(A)(s)$ for $s \in S_i$.

We provide a brief intuition here, utilizing what was observed in Section 2.4. Namely, that μ is associated with finite looping, and ν with infinite. So the odd equations make sure that when a run passes through odd parities it has to move back to a state with an even parity within a finite number of steps.

The equations do not just have to make sure that states with even parities are passed infinitely many times, but that the maximum parity occuring infinitely often is even. The proof presented in [9] is highly technical, focusing on algebraically analyzing the solutions of the system of equations. In the next section, we present an alternative proof using game semantics, providing a different perspective on the problem which we believe is more conceptual.

Combining Lemma 3.1 and Lemma 3.2 we obtain the coincidence result [9:Theorem 4.6]:

Theorem 3.3: Let $A = (S, \Sigma, \delta, s_0, \Sigma)$ be a parity tree automaton. Then the behavior mappings $\mathsf{beh}_1, ..., \mathsf{beh}_n$, which are the solution to the system of equations in Equation 20 coincide with the accepted language of A: $\mathsf{beh}(s_0) = [\mathsf{beh}_1, ..., \mathsf{beh}_n](s_0) = L(A)$.

4 Derivation of Coincidence Using Game Semantics

In this section we provide our derivation of the coincidence result Theorem 3.3. At the core of the derivation is Theorem 2.12, which relates a modal μ-calculus formula on a transition system and a parity game. We can apply Theorem 2.12 to derive the coincidence result with the following strategy:

1. Derive a closed μ-calculus formula from system of equations in Equation 20. Concretely, derive from a formula φ from the equations in Equation 20 a closed modal μ-calculus formula $\overline{\varphi}$ and define a transition system T_A from the parity tree automaton A such that we have $\tau \in \|\varphi\|(s)$, a tree is in the semantics of φ in state s, if and only if $(s, \tau) \in \|\varphi\|^{T_A}$, the formula $\overline{\varphi}$ holds in the state (s, τ) in the transition system T_A.
2. Apply Theorem 2.12 to conclude that $\overline{\varphi}$ holds in a state s if and only if there exists a winning strategy for V on $\mathcal{G}(T_A, \overline{\varphi})$ from state s.
3. Prove that there exists a winning strategy for V from state (s, w) in $\mathcal{G}(T_A, \overline{\varphi})$ if and only if $w \in L(A)(s)$

Unfortunately, we have not been able to prove step 3 for the parity acceptance criterion, just for the Büchi case. We do solve steps 1 and 2 for the general case (parity tree automata), and then provide the proof of step 3 for Büchi automata on trees (parity tree automata with two parities).

So the first step is defining the transition system from the parity tree automaton.

Definition 4.1: Let $A = (S, \Sigma, \delta, s_0, \Sigma)$ be a parity tree automaton, where we let $S = S_1 \cup ... \cup S_n$ the disjoint union of states with parity $1, ..., n$, respectively, Σ be the alphabet, and $\delta : S \times \Sigma \to \mathcal{P}(S^*)$ the transition function. We define a Transition System (TS) over the set of propositional variables $\{p_1, ..., p_n\}$ for this automaton, denoted as T_A, as follows:

- States are either (s, τ) for $s \in S$ and $\tau \in \mathrm{Tree}_\Sigma$ or $\left(\left(s_1, ..., s_{|\sigma|}\right), \left(\tau_1, ..., \tau_{|\sigma|}\right)\right)$ for $s_i \in S, \tau_i \in \mathrm{Tree}_\Sigma$
- Transitions from state (s, τ), for $s \in S, \sigma \in \Sigma, \tau \in \mathrm{Tree}_\Sigma$:

$$\left(s, \left(\sigma, \left(\tau_1, ..., \tau_{|\sigma|}\right)\right)\right) \to \left(\left(s'_1, ..., s'_{|\sigma|}\right), \left(\tau_1, ..., \tau_{|\sigma|}\right)\right) \text{ for all } \left(s'_1, ..., s'_{|\sigma|}\right) \in \delta(s)(\sigma) \tag{21}$$

- Transitions from state $\left(\left(s_1, ..., s_{|\sigma|}\right), \left(\tau_1, ..., \tau_{|\sigma|}\right)\right)$ for $s_i \in S, \sigma \in \Sigma, \tau_i \in \mathrm{Tree}_\Sigma$:

$$\left(\left(s_1, ..., s_{|\sigma|}\right), \left(\tau_1, ..., \tau_{|\sigma|}\right)\right) \to (s_i, \tau_i) \text{ for all } i \in \{1, ..., |\sigma|\} \tag{22}$$

- Labeling function given by $\lambda((s, \tau)) = \{p_i\}$ iff $s \in S_i$, i.e., the propositional variables denote for what i, we have $s \in S_i$.

By defining the states of the transition system as state-tree pairs on the parity tree automaton, we ensure that step 3 of the strategy succeeds: this setup allows for a clear correspondence between an infinite play in the parity game and an accepting run through the parity tree automaton.

Next, we derive a closed modal μ-calculus formula from the system of equations. Deriving the solution from the system of equations, as explained in Definition 2.9, obtains a closed formula. For example, for the Büchi case (two parities) the closed formula for $\mathsf{beh}_1 : S_1 \to \mathcal{P}(\mathrm{Tree}_\Sigma)$ from Equation 20 is $\mathsf{beh}_1 = \nu u_2.\Diamond_\delta[\mu u_1.[u_1, u_2] \upharpoonright S_1, u_2] \upharpoonright S_2$.

Note that this is not a modal μ-calculus formula, but a fixed point of an equation on the complete lattice $S \to \mathcal{P}(\Sigma^\omega)$. We want to define a modal μ-calculus formula $\overline{\varphi}$ and prove that the semantics of φ coincide with those of $\overline{\varphi}$ in the right way. To do so, we first observe how the closed formula φ is built up. Next, because the semantics of the formula φ are not explicitly given in [9], we define them concretely here.

We observe that the formula is built up inductively. If φ is a closed formula derived from the system of equations in Equation 20, then:

- $\varphi = U$ a free variable, or
- $\varphi = \Diamond_\delta \varphi'$, or
- $\varphi = \eta U.\varphi'$ where $\eta \in \{\mu, \nu\}$, or
- $\varphi = \varphi' \upharpoonright S_i$, or
- $\varphi = [\varphi_1, ..., \varphi_n]$

Next, we define the semantics for φ. For a valuation $V : Var \to (S \to \Sigma^\omega)$, the semantics of a formula $\|\varphi\|_V : S \to \mathcal{P}(\mathrm{Tree}_\Sigma)$ are:

- $\|U\|_V = V(U)$ for u a free variable,
- $\|\Diamond_\delta \varphi\|(s) = \left\{\left(\sigma, \left(\tau_1, ..., \tau_{|\sigma|}\right)\right) \mid \exists\left(s_1, ..., s_{|\sigma|}\right) \in \delta(s)(\sigma)[\forall i[\tau_i \in \|\varphi'\|_V(s_i)]]\right\}$,
- $\|\nu U.\varphi\| = \mathrm{lfp}\left(\lambda u.\ \|\varphi\|_{V[U \mapsto u]}\right)$ (note that the fixed points exists because $S \to \mathcal{P}(\mathrm{Tree}_\Sigma)$ is a complete lattice),
- $\|\mu U.\varphi\| = \mathrm{gfp}\left(\lambda u.\ \|\varphi\|_{V[U \mapsto u]}\right)$,
- $\|\varphi \upharpoonright S_i\|_V = \|\varphi\|_V \upharpoonright S_i$ (function restriction),
- $\|\varphi\|_V(s) = \begin{cases} \|\varphi_1\|_V(s) \text{ if } s \in S_1 \\ \vdots \\ \|\varphi_n\|_V(s) \text{ if } s \in S_n \end{cases}$

So we convert the closed formula from the system of equations to a modal μ-calculus formula and prove that the semantics coincide:

Definition 4.2: For a closed formula φ solution to Equation 20, we define the closed modal mu calculus formula $\overline{\varphi}$:

- $\varphi = u$ a free variable then $\overline{\varphi} = u$ also a free variable
- $\varphi = \Diamond_\delta \varphi'$ then $\overline{\varphi} = \Diamond\Box\overline{\varphi'}$
- $\varphi = \eta u.\varphi'$ for $\eta \in \{\mu, \nu\}$ then $\overline{\varphi} = \eta u.\overline{\varphi'}$

- $\varphi = \varphi' \upharpoonright S_i$ then $\overline{\varphi} = p_i \wedge \overline{\varphi'}$
- $\varphi = [\varphi_1, ..., \varphi_n]$ then $\overline{\varphi} = (p_1 \wedge \overline{\varphi_1}) \vee ... \vee (p_n \wedge \overline{\varphi_n})$

Lemma 4.3: For a closed formula φ solution to Equation 20 and a valuation $V : Var \to (S \to \mathcal{P}(\mathrm{Tree}_\Sigma))$, $s \in S, \tau \in \mathrm{Tree}_\Sigma$:

$$\tau \in \|\varphi\|_V\ (s) \Leftrightarrow (s, \tau) \in \|\overline{\varphi}\|_{\overline{V}}^{T_A} \tag{23}$$

where $\overline{V}(U) = \{(s, \tau) |\ s \in S, \tau \in V(U)(s)\}$

The proof is relatively straightforward by performing induction on the formula φ and can be found in Appendix A.2.

Next, we apply Theorem 2.12 to obtain a winning strategy for V on $\mathcal{G}(T_A, \overline{\varphi})$, so our final step is relating such a winning strategy with an accepting run on the Büchi automaton:

Lemma 4.4: For a closed formula φ solution to Equation 20, V has a winning strategy in the game $\mathcal{G}(\overline{\varphi}, T_A)$ from $(\overline{\varphi_i}, (s, \tau))$ iff the Büchi automaton A accepts the tree τ from s, i.e. $\tau \in L(A)(s)$.

The proof follows from a couple of key observations: the observation that the choices V makes when $\varphi = \Diamond\varphi'$ correspond to what transition to pick for the run ρ of A on τ; the transition R can pick when $\varphi = \Box\varphi'$ corresponds to checking that the maximum priority occuring infinitely often in every infinite branch is even. The concrete proof can be found in Appendix A.3. However, as noted before, the proof of this lemma is only for the Büchi case (two parities). The hard step for higher priorities is reasoning about parity of the subformulas of the closed formula from the system of equations because they get big and convoluted quickly. The parity then depends on the alternation depth of these formulas (see Definition 2.11) which is hard for these large and complicated formulas.

The proof of Theorem 3.3 (for Büchi automata on trees) now follows from Lemma 3.1, Lemma 4.3, Theorem 2.12 and Lemma 4.4.

5 Conclusion and Future Work

In this report we have shown a coalgebraic representation of Büchi automata, and more generally, parity tree automata [9]. The construction relies upon two key ideas: working in the Kleisli category for the monad $\mathcal{P}$ and deriving separate commuting diagrams for states with different parities and then utilizing fixed point equations for these different mappings for each parity.

We explained the model in the Kleisli category in Section 3.1 by showing how to construct a final coalgebra for finite words for a nondeterministic automaton. Subsequently we constructed a weakly final coalgebra to additionally obtain the infinite words within such a system. Building upon these ideas we derived the coalgebraic construction for parity tree automata in Section 3.2, making use of the modal μ-calculus explained in Section 2.4.

In Section 4 we presented our alternate derivation of the coincidence results in Section 3.2. By applying game semantics we were able to give a more conceptual proof of the results. Additionally, looking at this result through this alternate angle of game semantics coul provide new insights.

Seeing if this alternate derivation can shed new light onto the topic, for example connections to coalgebra automata which are based on game theoretic techniques [6], is one direction of future work that could build upon this report. Secondly, future work could look into proving Lemma 4.4 for parity automata. As explained in Section 4, we were unable to prove this lemma for the most general case due to the complicated nature of the system of equations. Bridging this gap would complete the alternate derivation of the coincidence result for all parity tree automata.

Bibliography

[1] André Arnold and Damian Niwinski. 2001. *Rudiments of mu-calculus*. Elsevier.

[2] Steve Awodey. 2010. *Category theory*. OUP Oxford.

[3] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. 2003. *Automata, logics, and infinite games: a guide to current research.* Springer.

[4] Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. 2007. Generic trace semantics via coinduction. *Logical Methods in Computer Science* 3, (2007).

[5] Bart Jacobs. 2004. Trace semantics for coalgebras. *Electronic Notes in Theoretical Computer Science* 106, (2004), 167–184.

[6] Clemens Kupke and Yde Venema. 2008. Coalgebraic automata theory: basic results. *Logical Methods in Computer Science* 4, (2008).

[7] John C Martin. 1991. *Introduction to Languages and the Theory of Computation.* McGraw-Hill NY.

[8] Jan JMM Rutten. 2000. Universal coalgebra: a theory of systems. *Theoretical computer science* 249, 1 (2000), 3–80.

[9] Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. 2016. Coalgebraic Trace Semantics for Buechi and Parity Automata. In *27th International Conference on Concurrency Theory (CONCUR 2016)*, 2016. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 1–15.

[10] Moshe Y. Vardi. 1996. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, Faron Moller and Graham Birtwistle (eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 238–266. https://doi.org/10.1007/3-540-60915-6_6

A Proofs

A.1 *Proof of Lemma 3.1*

First we unfold some definitions:

$\left(\eta_{\mathrm{Tree}_\Sigma} d\right)^{-1} = \eta_{\mathrm{Tree}_\Sigma}(d^{-1})$ where $d^{-1}\left(\tau_1, ..., \tau_{|\sigma|}\right) = \left(\sigma, \left(\tau_1, ..., \tau_{|\sigma|}\right)\right) \in \mathrm{Tree}_\Sigma$.

Let us call $u_1 + ... + u_n = \mathsf{beh}$ such that $\overline{F}[u_1, ..., u_n] = \overline{F}(\mathsf{beh}) = \lambda_{\mathrm{Tree}_\Sigma} \circ \left(\sqcup_{\sigma\in\Sigma} \mathsf{beh}^{|\sigma|}\right)$ so and see that $\sqcup_{\sigma\in\Sigma} \mathsf{beh}^{|\sigma|} : \left(\sqcup_{\sigma\in\Sigma} S^{|\sigma|}\right) \to \left(\sqcup_{\sigma\in\Sigma} \left(\mathcal{P}(\mathrm{Tree}_\Sigma)\right)^{|\sigma|}\right)$, maps a tuple of states $\left(s_1, ..., s_{|\sigma|}\right)$ to $\left(\mathsf{beh}(s_1), ..., \mathsf{beh}\left(s_{|\sigma|}\right)\right)$, i.e. to the languages accepted by every state s_i. Combining with the natural transformation $\lambda : \left(\sqcup_{\sigma\in\Sigma} \left(\mathcal{P}(\mathrm{Tree}_\Sigma)\right)^{|\sigma|}\right) \to \mathcal{P}\left(\sqcup_{\sigma\in\Sigma} \mathrm{Tree}_\Sigma^{|\sigma|}\right)$ defined by $\lambda\left(\mathrm{T}_1, ..., \mathrm{T}_{|\sigma|}\right) = \left\{\left(\tau_1, ..., \tau_{|\sigma|}\right) |\ \tau_i \in \mathrm{T}_i\right\}$ we get $\overline{F}[u_1, ..., u_n]\left(s_1, ..., s_{|\sigma|}\right) = \left\{\left(\tau_1, ..., \tau_{|\sigma|}\right) \mid \tau_i \in \mathsf{beh}(s_i)\right\}$

$c_i = c \circ \kappa_i : S_i \to \mathcal{P}\left(\sqcup_{\sigma\in\Sigma} S^{|\sigma|}\right)$ in terms of the automaton is defined as $c_i(s) = \left\{S' |\ \sigma \in \Sigma, S' = \left(s'_1, ..., s'_{|\sigma|}\right) \in \delta(s)(\sigma)\right\} \in \mathcal{P}\left(\sqcup_{\sigma\in\Sigma} S^{|\sigma|}\right)$ for $s \in S_i$, i.e., tuples of succesor states for each σ.

Combining these, and writing out the Kleisli composition in terms of functions in **Sets** we get:

$$\left(\eta_{\mathrm{Tree}_\Sigma} d\right)^{-1} \odot \overline{F}[u_1, ..., u_n] \odot c_i = \mu_{\mathrm{Tree}_\Sigma} \circ \mathcal{P}\left(\eta_{\mathrm{Tree}_\Sigma} \circ d^{-1}\right) \circ \left(\mu_{\sqcup_{\sigma\in\Sigma}\mathrm{Tree}_\Sigma^{|\sigma|}} \circ \mathcal{P}\left(\lambda \circ \sqcup_{\sigma\in\Sigma} \mathsf{beh}^{|\sigma|}\right) \circ c_i\right) \tag{24}$$

Observing that $\mu_{\mathrm{Tree}_\Sigma} \circ \mathcal{P}\left(\eta_{\mathrm{Tree}_\Sigma} \circ d^{-1}\right) = \mathcal{P}(d^{-1})$), letting $u_1 + ... + u_n = \mathsf{beh}$ again and combining $\mathcal{P}\left(\lambda \circ \left(\sqcup_{\sigma\in\Sigma} \mathsf{beh}^{|\sigma|}\right)\right)$ and c_i by using our observations from above we obtain, for an $s \in S_i$:

$$\begin{aligned} &\mu_{\mathrm{Tree}_\Sigma} \circ \mathcal{P}\left(\eta_{\mathrm{Tree}_\Sigma} \circ d^{-1}\right) \circ \left(\mu_{\sqcup_{\sigma\in\Sigma}\mathrm{Tree}_\Sigma^{|\sigma|}} \circ \mathcal{P}\left(\lambda \circ \sqcup_{\sigma\in\Sigma} \mathsf{beh}^{|\sigma|}\right) \circ c_i\right)(s) \\ &= \mathcal{P}(d^{-1})\left(\left\{\left(\tau_1, ..., \tau_{|\sigma|}\right) \mid \sigma \in \Sigma, \left(s'_1, ..., s'_{|\sigma}\right) \in \delta(s)(\sigma), \tau_i \in [\mathsf{beh}](s'_i)\right\}\right) \\ &= \left\{\left(\sigma, \left(\tau_1, ..., \tau_{|\sigma|}\right)\right) \mid \sigma \in \Sigma, \left(s'_1, ..., s'_{|\sigma}\right) \in \delta(s)(\sigma), \tau_i \in [\mathsf{beh}](s'_i)\right\} = \Diamond_\delta(\mathsf{beh})(s) \end{aligned} \tag{25}$$

□

A.2 *Proof of Lemma 4.3*

We prove this by induction on the formula φ. The base case is $\varphi = U$ a free variable:

$\tau \in \|U\|_V(s) = V(U)(s) \leftrightarrow (s, \tau) \in \overline{V}(U) = \|U\|_{\overline{V}}^{T_A}$

Induction step:

- $\varphi = \mu U.\varphi'$:

We have to show $\tau \in \|\mu U.\varphi'\|_V(s) = \text{lfp}\big(\lambda u.\ \|\varphi'\|_{V[U\mapsto u]}\big) \Leftrightarrow (s,\tau) \in \|\mu U.\overline{\varphi'}\|_{\overline{V}} = \text{lfp}\ \big(\lambda u.\ \|\overline{\varphi'}\|_{\overline{V}[U\mapsto u]}\big)$. Let $W = \text{lfp}\big(\lambda u.\ \|\varphi'\|_{V[U\mapsto u]}\big)$. We define $\overline{W} = \{(s,\tau) \mid s \in S, \tau \in W(s)\}$ and show $W = \text{lfp}\big(\lambda u.\ \|\varphi'\|_{V[U\mapsto u]}\big) \Leftrightarrow \overline{W} = \text{lfp}\big(\lambda u.\ \|\overline{\varphi'}\|_{V[U\mapsto u]}\big)$. For this we first prove that W is a fixed point iff $\overline{W}$ is a fixed point:

Assume W is a fixed point, so $\|\varphi'\|_{V[U\mapsto W]} = W$. We observe that for a valuation V and V' where $V' = V[U \mapsto W]$, we have the converted valuation $\overline{V'} = \overline{V}\big[U \mapsto \overline{W}\big]$. We use this to incite the IH to get $w \in \|\varphi'\|_{V[U\mapsto W]} \Leftrightarrow (s,\tau) \in \|\overline{\varphi'}\|_{\overline{V}[U\mapsto\overline{W}]}$. Using this we get $(s,\tau) \in \|\overline{\varphi'}\|_{\overline{V}[U\mapsto\overline{W}]} \Leftrightarrow w \in \|\varphi'\|_{V[U\mapsto W]}(s) = W(s) \Leftrightarrow (s,\tau) \in \overline{W}$, so $\|\overline{\varphi'}\|_{\overline{V}[U\mapsto\overline{W}]} = \overline{W}$, so $\overline{W}$ is a fixed point.

Now assume $\overline{W}$ is a fixed point, so $\|\overline{\varphi'}\|_{\overline{V}[U\mapsto\overline{W}]} = \overline{W}$. Then, for $s \in S$, $W(s) = \big\{s \mid (s,\tau) \in \overline{W}\big\}$. Applying IH like the previous case again we obtain $w \in \|\varphi'\|_{V[U\mapsto W]}(s) \Leftrightarrow (\tau,s) \in \|\overline{\varphi'}\|_{\overline{V}[U\mapsto\overline{W}]} = W \Leftrightarrow \tau \in W(s)$. So $w \in \|\varphi'\|_{V[U\mapsto W]}(s) \Leftrightarrow \tau \in W(s)$ for all $s \in S$, so $\|\varphi'\|_{V[U\mapsto W]} = W$, so W is a fixed point.

Next, we show that W is the *least* fixed point iff $\overline{W}$ is the *least* fixed point:

Assume W is a lfp, from above we know that $\overline{W}$ is a fixed point. Take some other fixed point $\overline{Y}$, i.e. $\|\overline{\varphi'}\|_{\overline{V}[U\mapsto\overline{Y}]} = \overline{Y}$. Now, again inciting what we showed above, we know Y is a fixed point, so $\|\varphi'\|_{V[U\mapsto\overline{Y}]} = \overline{Y}$. So because W is the lfp, for all s, $W(s) \subseteq Y(s)$. From this it follows that $(s,\tau) \in \overline{W} \to \tau \in W(s) \to \tau \in Y(s) \to (s,\tau) \in \overline{Y}$, so $\overline{W} \subseteq \overline{Y}$. So $\overline{W}$ is the least fixed point.

For the other direction, assume $\overline{W}$ is a least fixed point. Then W is a fixed point. Take some other fixed point Y, i.e. $\|\varphi'\|_{V[U\mapsto Y]} = Y$, then $\overline{Y}$ is a fixed point. So because $\overline{W}$ is the lfp, we have $\overline{W} \subseteq \overline{Y}$. Now for any τ, s we have $\tau \in W(s) \to (s,\tau) \in \overline{W} \to (s,\tau) \in \overline{Y} \to \tau \in Y(s)$. So $W \subseteq Y$.

- $\varphi = \nu U.\varphi'$:

This case is a analagous to the μ case. The first part proving W is a fixed point iff $\overline{W}$ is a fixed point, and for proving W is a *greatest* fixed point iff $\overline{W}$ is too you reason in the opposite direction as for μ.

- $\varphi = \Diamond_\delta \varphi'$:

$$
\begin{aligned}
&\tau \in \|\Diamond_\delta \varphi'\|_V(s) \\
&\quad = \Big\{\big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big) \mid \exists\big(s_1, ..., s_{|\sigma|}\big) \in \delta(s)(\sigma)[\forall i[\tau_i \in \|\varphi'\|_V(s_i)]]\Big\} \\
&\quad \overset{IH}{=} \Big\{\big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big) \mid \exists\big(s_1, ..., s_{|\sigma|}\big) \in \delta(s)(\sigma)\big[\forall i\big[(s_i, \tau_i) \in \|\overline{\varphi'}\|_{\overline{V}}\big]\big]\Big\} \\
&\quad \overset{*}{=} \Big\{\big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big) \mid \exists\big(s_1, ..., s_{|\sigma|}\big) \in \delta(s)(\sigma)\Big[\Big(\big(s_1, ..., s_{|\sigma|}\big), \big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big)\Big) \in \|\Box\overline{\varphi'}\|_{\overline{V}}\Big]\Big\} \\
&\quad \leftrightarrow (s,\tau) \in \Big\{\big(s, \big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big)\big) \mid \exists\big(s_1, ..., s_{|\sigma|}\big) \in \delta(s)(\sigma)\Big[\Big(\big(s_1, ..., s_{|\sigma|}\big), \big(\sigma, \big(\tau_1, ..., \tau_{|\sigma|}\big)\big)\Big) \in \|\Box\overline{\varphi'}\|_{\overline{V}}\Big]\Big\} \\
&\quad \overset{**}{=} \|\Diamond\Box\overline{\varphi'}\|_{\overline{V}}^{T_A}
\end{aligned}
\tag{26}
$$

- $\varphi = \varphi' \upharpoonright S_i$:

$\tau \in \|\varphi' \upharpoonright S_i\|_V(s) \leftrightarrow s \in S_i \wedge \tau \in \|\varphi'\|_V(s) \overset{IH}{\leftrightarrow} s \in S_i \wedge (s,\tau) \in \|\overline{\varphi'}\|_{\overline{V}} \leftrightarrow (s,\tau) \in \|p_i \wedge \overline{\varphi'}\|_{\overline{V}}$

- $\varphi = [\varphi_1, ..., \varphi_n]$:

$\|\varphi\|_V(s) = \begin{cases} \|\varphi_1\|_V(s) \text{ if } s \in S_1 \\ \vdots \\ \|\varphi_n\|_V(s) \text{ if } s \in S_n \end{cases}$, so let $\tau \in \|\varphi\|_V(s)$ for $s \in S_i$, then $\tau \in \|\varphi_i\|_V(s)$ so by IH $(s,\tau) \in \|\overline{\varphi_i}\|_{\overline{V}}(s)$, and because $s \in S_i$, $(s,\tau) \in S_i$, $(s,\tau) \in \|p_i \wedge \overline{\varphi_i}\|_{\overline{V}}(s)$ and thus $(s,\tau) \in \|(p_1 \wedge \varphi_1) \vee ... \vee (p_n \wedge \varphi_n)\|_{\overline{V}} = \|\overline{\varphi}\|_{\overline{V}}$.

Now $(s,\tau) \in \|\overline{\varphi}\|_{\overline{V}} = \|(p_1 \wedge \overline{\varphi_1}) \vee ... \vee (p_n \wedge \overline{\varphi_n})\|_{\overline{V}}$. Take i such that $(s,\tau) \in \|p_i \wedge \overline{\varphi_i}\|_{\overline{V}}$ then we have $s \in S_i$ and (by IH) $w \in \|\varphi_i\|_V(s)$, and by definition of $\|[\varphi_1, ..., \varphi_n]\|$ then $\tau \in \|\varphi\|_V(s)$.

A.3 *Proof of Lemma 4.4*

Observe the closed formulas for Equation 20 l^1_{sol} and l^2_{sol}:

- $l^1_{\text{sol}} = \mu u_1.\Diamond^1_\delta\big[u_1, \nu u_2.\Diamond^2_\delta\big[\big(\mu u_1'.\Diamond^1_\delta[u_1', u_2]\big), u_2\big]\big)$,

so $\overline{\varphi_1} = \mu u_1.p_1 \wedge \Diamond((p_1 \wedge u_1) \vee (p_2 \wedge \nu u_2.(p_2 \wedge \Diamond((\mu u_1'.p_1 \wedge \Diamond((p_1 \wedge u_1') \vee (p_2 \wedge u_2))) \vee (p_2 \wedge u_2)))))$. We observe that $\Omega(\mu u_1....) = 1$, $\Omega(\nu u_2....) = 2$ and $\Omega(\mu u_1'....) = 1$.

- $l^2_{\text{sol}} = \nu u_2.\Diamond^2_\delta\left[\left(\mu u_1'.\Diamond^1_\delta[u_1', u_2]\right), u_2\right]$

  so $\overline{\varphi_2} = \nu u_2.(p_2 \wedge \Diamond((\mu u_1'.p_1 \wedge \Diamond((p_1 \wedge u_1') \vee (p_2 \wedge u_2))) \vee (p_2 \wedge u_2))))$. We observe that $\Omega(\mu u_2....) = 2$, $\Omega(\nu u_1....) = 1$.

Assume player V has a winning strategy for $\mathcal{G}(\overline{\varphi_i}, T_A)$ from $(\overline{\varphi_i}, (s, \tau))$. Observe that through V's choices when $\varphi = \Diamond\varphi', (\Diamond\varphi', \left(s, \left(\sigma, \left(\tau_1, ..., \tau_{|\sigma|}\right)\right)\right) \to \left(\varphi', \left((s_1', ..., s_n'), \left(\tau_1, ..., \tau_{|\sigma|}\right)\right)\right)$ tell how to construct the run ρ on A. Because V has a winning strategy, R can pick whatever transition it wants when $\varphi = \Box\varphi'$ and the play will be won by V, which means the maximal priority of the states in $\mathcal{G}(\overline{\varphi_i}, T_A)$ is even. This corresponds to checking an infinite branch of the run ρ of A on τ and because the even parity in the game corresponds to S_2 in the automaton, the maximum priority of the states in A occuring infinitely often in every infinite branch of the run ρ of A on τ is even, which means τ is accepted by T from state s.

The other way is similar. We again osbserve that the choice V has to make when $\varphi = \Diamond\varphi'$ is decided by the run of A on τ, and the fact that regardless of the transitions R picks when $\varphi = \Box\varphi'$ the play is winning by V follows from the fact that for every infinite branch of the run the maximum priority of the states occuring infinitely often is even, which corresponds to the even parities in the game, and thus the play in the game is winning by V. Furthermore, note that the choice to pick at $(((p_1 \wedge \overline{\varphi_1}) \vee ... \vee (p_n \wedge \overline{\varphi_n})), (s, \tau))$ simply depends on i such that $s \in S_i$, and that also makes sure that R cannot pick the left formula in the formula $p_i \wedge \overline{\varphi_i}$ to win that way becasue $s \in S_i$.