

# On-the-fly Algorithms for GKAT Equivalences

Cheng Zhang<sup>§</sup>

Department of Computer Science  
University College London  
London, United Kingdom  
0000-0002-8197-6181

Qiancheng Fu

Department of Computer Science  
Boston University  
Boston, USA  
email address or ORCID

Hang Ji

Department of Computer Science  
Boston University  
Boston, USA  
email address or ORCID

Ines Santacruz

Department of Computer Science  
Boston University  
Boston, USA  
email address or ORCID

Marco Gaboardi

Department of Computer Science  
Boston University  
Boston, USA  
email address or ORCID

**Abstract**—This document is a model and instructions for  $\text{\LaTeX}$ . This and the `IEEEtran.cls` file define the components of your paper [title, text, heads, etc.]. **\*CRITICAL: Do Not Use Symbols, Special Characters, Footnotes, or Math in Paper Title or Abstract.**

**Index Terms**—component, formatting, style, styling, insert.

## I. INTRODUCTION

a) *Notation*:: In this paper, we will use uncurried notation to apply curried functions, for example, given a function  $\delta : X \rightarrow Y \rightarrow Z$ , we will write the function applications as  $\delta(x) : Y \rightarrow Z$  and  $\delta(x, y) : Z$ .

## II. BACKGROUND ON COALGEBRA AND GKAT

### A. Concepts in Universal Coalgebra

In this paper, we will make heavy use of coalgebraic theory, thus it is empirical for us to recall some notions and useful theorems in universal coalgebra. Given a functor  $F$  on the category of set and functions, a *coalgebra over  $F$*  or  *$F$ -coalgebra* consists of a set  $S$  and a function  $\sigma_S : S \rightarrow F(S)$ . We typically call elements in  $S$  the *states* of the coalgebra, and  $\sigma_S(s)$  the *dynamic* of state  $s$ . We will sometimes use the states  $S$  to denote the coalgebra, when no ambiguity can arise.

A homomorphism between two  $F$ -coalgebra  $S$  and  $U$  is a map  $h : S \rightarrow U$  that preserves the function  $\sigma$ ; diagrammatically, the following diagram commutes:

$$\begin{array}{ccc} S & \xrightarrow{h} & U \\ \sigma_S \downarrow & & \downarrow \sigma_U \\ F(S) & \xrightarrow{F(h)} & F(U) \end{array}$$

Identify applicable funding agency here. If none, delete this.

<sup>§</sup>Work largely performed at Boston University

When we can restrict the homomorphism map into an inclusion map  $i : S' \rightarrow S$  for  $S' \subseteq S$  then we say that  $S'$  is a *sub-coalgebra* of  $S$ , denoted as  $S' \sqsubseteq S$ . Specifically, the following diagram commutes when  $S' \sqsubseteq S$ :

$$\begin{array}{ccc} S' & \xhookrightarrow{i} & S \\ \sigma_{S'} \downarrow & & \downarrow \sigma_S \\ F(S') & \xhookrightarrow{F(i)} & F(S) \end{array}$$

In fact, the function  $\sigma_{S'}$  is uniquely determined by the states  $S'$  [6, Proposition 6.1]. Sub-coalgebras are also preserved under homomorphic images and pre-images:

**Lemma 1** (Theorem 6.3 [6]). *given a homomorphism  $h : S \rightarrow U$ , and sub-coalgebras  $S' \sqsubseteq S$  and  $U' \sqsubseteq U$ , then*

$$h(S') \sqsubseteq U \text{ and } h^{-1}(U') \sqsubseteq S.$$

One particularly important sub-coalgebra of a coalgebra  $S$  is the least sub-coalgebra that contains a state  $s$ . We will denote this sub-coalgebra as  $\langle s \rangle_S$ , and call it *principle sub-coalgebra* generated by  $s$ . We sometimes omit the subscript  $S$  when it can be inferred from context or irrelevant. Intuitively, we usually think of principle sub-coalgebra  $\langle s \rangle_S$  as the sub-coalgebra that is formed by all the “reachable state” from  $s$ . This coalgebraic characterization of reachable state can allow us to avoid induction on the length of path from  $s$  to a state in  $\langle s \rangle_S$ .

For all coalgebra  $S$  and a state  $s \in S$ , principle sub-coalgebra  $\langle s \rangle_S$  always exists and is unique, because sub-coalgebra of any coalgebra forms a complete lattice [6, theorem 6.4]; thus taking the meet of all the sub-coalgebra that contains  $s$  will yield  $\langle s \rangle_S$ .

Similar to sub-coalgebra, principle sub-coalgebra is also preserved under homomorphic image:

**Theorem 2.** *Homomorphic image preserves principle sub-GKAT coalgebra. Specifically, given a homomorphism  $h : S \rightarrow U$ :*

$$h(\langle s \rangle_S) = \langle h(s) \rangle_U$$

*Proof.* We will need to show that  $h(\langle s \rangle_S)$  is the smallest sub-GKAT coalgebra of  $S$  that contain  $h(s)$ . By definition of image,  $h(\langle s \rangle_S)$  indeed contain  $h(s)$ . By Lemma 1,  $h(\langle s \rangle_S) \sqsubseteq U$ , i.e.  $h(\langle s \rangle_S)$  is a sub-coalgebra of  $U$ . Finally, take any sub-coalgebra  $U' \sqsubseteq U$  s.t.  $h(s) \in U'$ :

$$\begin{aligned} h(s) \in U' &\implies s \in h^{-1}(U') \\ &\implies \langle s \rangle_S \sqsubseteq h^{-1}(U') \quad \text{definition of } \langle s \rangle_S \\ &\implies h(\langle s \rangle_S) \sqsubseteq U' \quad \text{Lemma 1} \end{aligned}$$

Hence  $h(\langle s \rangle_S)$  is the smallest sub-GKAT coalgebra of  $U$  that contains  $h(s)$ .  $\square$

A *final coalgebra*  $\mathcal{F}$  over a signature  $F$ , sometimes called the *behavior* or *semantics* of coalgebras over  $F$ , is an  $F$ -coalgebra s.t. for all  $F$ -coalgebra  $S$ , there exists a unique homomorphism  $\text{beh}_S : S \rightarrow \mathcal{F}$ .

Given two  $F$ -coalgebra  $S$  and  $U$ , the *behavioral equivalence* between states in  $S$  and  $U$  can be computed by a notion called *bisimulation*. A relation  $\sim \subseteq S \times U$  is called a *bisimulation relation* if it forms an  $F$ -coalgebra:

$$\sigma_\sim : \sim \rightarrow F(\sim),$$

and its projections  $\pi_1 : \sim \rightarrow S$  and  $\pi_2 : \sim \rightarrow U$  are both homomorphisms:

$$\begin{array}{ccccc} S & \xleftarrow{\pi_1} & \sim & \xrightarrow{\pi_2} & U \\ \sigma_S \downarrow & & \downarrow \sigma_\sim & & \downarrow \sigma_U \\ F(S) & \xleftarrow{F(\pi_1)} & F(\sim) & \xrightarrow{F(\pi_2)} & F(U) \end{array}$$

In a special case, when there exists a homomorphism  $h : S \rightarrow U$ , then we can simply pick  $\sim \subseteq S \times U$  to be  $\{(s, h(s)) \mid s \in S\}$ , which gives us a bisimulation with the lift  $\sigma_\sim \triangleq \sigma_S \times \sigma_U$ .

When given a homomorphism  $h : S \rightarrow U$ , we can construct a bisimulation  $\sim \subseteq S \times U$ , where it relates all the states  $s \in S$  with  $h(s) \in U$ ; and each component of the pair will transition individually via their respective transition function  $\delta_\sim \triangleq \delta_S \times \delta_U$ , where  $\times$  is the bifunctorial lift of product.

**Corollary 3.** *Given a homomorphism  $h : S \rightarrow U$  between two  $F$ -coalgebra  $S, U$ , then all for all states  $s \in S$ ,  $\text{beh}_S(s) = \text{beh}_U(h(s))$ .*

## B. GKAT and Its Coalgebra

Guarded Kleene Algebra with Tests, or GKAT [8], is a deterministic fragment of Kleene Algebra with Tests. The syntax of GKAT over a set of primitive actions  $K$  and a set of primitive tests  $T$  can be defined in two sorts, boolean expressions  $\text{BExp}$  and GKAT expressions  $\text{Exp}$ :

$$\begin{aligned} a, b, c &\in \text{BExp} \triangleq 1 \mid 0 \mid t \in T \mid b \wedge c \mid b \vee c \mid \bar{b} \\ e, f &\in \text{Exp} \triangleq p \in K \mid b \in \text{BExp} \mid e +_b f \mid e; f \mid e^{(b)} \end{aligned}$$

where  $e +_b f$  is the if statement with condition  $b$ ,  $e; f$  is the sequencing of expression  $e$  and  $f$ , and  $e^{(b)}$  is the while loop with body  $e$  and condition  $b$ . We use notation like  $b \leq c$ ,  $b \equiv c$ , and  $b \not\equiv c$  for the usual order, equivalence, and inequivalence in Boolean Algebra. A GKAT expression can be unfolded into a KAT expression in the usual manner [4]:

$$e +_b f \triangleq b; e + \bar{b}; f \quad e^{(b)} \triangleq (b; e)^*; \bar{b}.$$

Then the semantics of each expression  $\llbracket e \rrbracket$  can be computed by the semantics of Kleene Algebra with tests [4]. An important construct in the semantics is *atoms*, which are conjunctions of all the primitive tests either in its positive or negative form: for  $T \triangleq \{t_1, t_2, \dots, t_n\}$

$$\mathbf{At}_T \triangleq \{t'_1 \wedge t'_2 \wedge \dots \wedge t'_n \mid t'_i \in \{t_i, \bar{t}_i\}\}.$$

Follow the conventional notation, we denote an atom using  $\alpha, \beta$ ; and we sometimes omit the subscript  $T$  when no confusing can arise. Alternatively, atoms can also be thought of as truth assignments to each primitive tests, indicating which primitive tests is satisfied in the current program states; and  $\alpha \leq b$  if and only if the truth assignment represented by  $\alpha$  satisfies  $b$ .

For the sake of brevity, we omit the complete definition of GKAT and KAT semantics, we refer the reader to previous works [8, 7, 4], which explains these semantics in detail. Our work avoids direct interaction with the semantics by leveraging prior results in the coalgebraic theory of GKAT, which we will recap below.

Formally, GKAT coalgebras over primitive actions  $K$  and primitive tests  $T$  are coalgebras over the following functor:

$$G(S) \triangleq (\{\text{acc}, \text{rej}\} + S \times K)^{\mathbf{At}_T}.$$

Intuitively, given a state  $s \in S$  and an atom  $\alpha \in \mathbf{At}$ ,  $\delta(s, \alpha)$  will deterministically execute one of the following: reject  $\alpha$  when  $\delta(s, \alpha) = \text{rej}$ ; accept  $\alpha$  when  $\delta(s, \alpha) = \text{acc}$ ; or transition to a state  $s' \in S$  and execute action  $p \in K$  when  $\delta(s, \alpha) = (s', p)$ .

In particular,  $G$  is a simple polynomial functor [2], allowing the sub-coalgebra to preserve and reflect bisimulations.

**Theorem 4** (sub-coalgebras preserve and reflect bisimulation). *Given two states in sub-coalgebra  $s \in S' \subseteq S$  and  $u \in U' \subseteq U$ , there exists a bisimulation  $\sim' \subseteq S' \times U'$  s.t.  $s \sim' u$  if and only if there exists a bisimulation  $\sim \subseteq S \times U$  s.t.  $s \sim u$ .*

*Proof.* For  $\Rightarrow$  direction, we will show that if  $\sim' \subseteq S' \times U'$  is a bisimulation, then  $\sim'$  is also a bisimulation between  $S$  and  $U$ :

$$\begin{array}{ccccccc} S & \xleftarrow{i} & S' & \xleftarrow{\pi_1} & \sim & \xrightarrow{\pi_2} & U' \xleftarrow{i} U \\ \downarrow \delta_S & & \downarrow \delta_{S'} & & \downarrow \delta_{\sim} & & \downarrow \delta_{S'} \downarrow \delta_U \\ G(S) & \xleftarrow{G(i)} & G(S') & \xleftarrow{G(\pi_1)} & G(\sim) & \xrightarrow{G(\pi_2)} & U' \xleftarrow{G(i)} U \end{array}$$

Because the inclusion homomorphism  $i$  doesn't change the input, we have:

$$\begin{aligned} \sim & \xrightarrow{\pi_1} S' \xrightarrow{i} S = \sim \xrightarrow{\pi_1} S; \\ \sim & \xrightarrow{\pi_2} U' \xrightarrow{i} U = \sim \xrightarrow{\pi_2} U. \end{aligned}$$

To prove the  $\Leftarrow$  we will show that if  $\sim \subseteq S \times U$  is a bisimulation, then the restriction  $\sim' \triangleq \{(s, u) \in \sim \mid s \in S', u \in U'\}$  is a bisimulation between  $S'$  and  $U'$ . We first realize that  $\sim_{S', U'}$  is a pre-image of the maximal bisimulation  $\equiv_{S', U'}$  along the inclusion homomorphism  $i : \sim \rightarrow \equiv_{S, U}$ . This means that  $\sim_{S', U'}$  can be formed by a pullback square:

$$\begin{array}{ccc} \sim_{S', U'} & \xrightarrow{i} & \equiv_{S', U'} \\ \downarrow i & \lrcorner & \downarrow i \\ \sim & \xrightarrow{i} & \equiv_{S, U} \end{array}$$

Recall that elementary polynomial functor [2] like  $G$  preserves pullback, hence the pullback also uniquely generates a GKAT coalgebra [6]  $\square$

Besides nice property with bisimulation, GKAT coalgebra is also deterministic, unlike Kleene coalgebra with tests (KCT) [3]. For each atom, states in a KCT can accept or reject the atom (but not both), yet states can also non-deterministically transition to multiple different states via the same atom, while executing different actions. As we will see later, the deterministic behavior of GKAT coalgebra not only enables a more versatile symbolic algorithm than KCT [5], but also present challenges. Specifically, GKAT coalgebra requires normalization to compute finite trace equivalences [8], where we reroute all the transitions that cannot lead to acceptance immediately into rejection; and states that can never lead to acceptance is called *dead states*.

In previous works, these dead states are detected after the necessary coalgebra is computed and stored. This approach requires storing all the transition and states of coalgebra in memory, meaning that the algorithm does not short circuit even if a mismatch can be detected in the starting state; for example, when for some  $\alpha \in \mathbf{At}$ ,  $\delta_S(s, \alpha) = \text{rej}$  and  $\delta_U(u, \alpha) = \text{acc}$ .

Our algorithm is lazy in the dead state detection i.e. the dead state are only checked when a mismatch requires it; even then, our dead state detection algorithm will only check the necessary states to compute the liveness of the states causing the mismatch. This not only avoids unnecessarily checking the liveness of a state, but also enables allows on-the-fly generation of the coalgebra, like using derivatives, where we can remove irrelevant states from memory after it has been explored.

However, to truly understand our on-the-fly algorithms, we will first need to define “dead states”, and its role in defining the coalgebraic semantic of GKAT.

### C. Liveness and Sub-GKAT coalgebras

Traditionally, live and dead states are defined by whether they can reach an accepting state [8]. However, we can use induction on the length of the path to show that principle sub-coalgebra  $\langle s \rangle_S$  contains exactly the reachable states of  $s$  in any GKAT coalgebra  $S$ . Thus, the classical definition is equivalent to the following:

**Definition 1** (liveness of states). *A state  $s$  is accepting if there exists an atom  $\alpha \in \mathbf{At}$  s.t.  $\delta(s, \alpha) = \text{acc}$ ; a state  $s'$  is live if there exists an accepting state  $s' \in \langle s \rangle$ ; and a state  $s'$  is dead if there is no accepting state in  $\langle s \rangle$ .*

This alternative liveness definition can help us prove important theorems regarding reachability and liveness without explicitly performing induction on traces. We show the following theorems as examples:

**Lemma 5.** *A state  $s$  is dead if and only if all elements in  $\langle s \rangle$  is dead.*

*Proof.*  $\Leftarrow$  direction is true, because  $s \in \langle s \rangle$ : if all  $\langle s \rangle$  is dead, then  $s$  is dead.  $\Rightarrow$  direction can be proven as follows. Take any  $s' \in \langle s \rangle$ , then  $\langle s' \rangle \subseteq \langle s \rangle$  because  $s'$  is the minimal sub-coalgebra that contains  $s'$ . Since there is no accepting state in  $\langle s \rangle$ , thus there cannot be any accepting state in  $\langle s' \rangle$ , hence  $s'$  is also dead.  $\square$

**Theorem 6** (homomorphism preserves liveness). *Given a homomorphism  $h : S \rightarrow U$  and a state  $s \in S$ :*

$$s \text{ is live} \iff h(s) \text{ is live}$$

*Proof.* Because homomorphic image preserves principle sub-GKAT coalgebra (Theorem 2)

$$h(\langle s \rangle_S) = \langle h(s) \rangle_U;$$

therefore for any state  $s' \in S$ :

$$s' \in \langle s \rangle_S \iff h(s') \in h(\langle s \rangle_S) \iff h(s') \in \langle h(s) \rangle_U.$$

And because  $s'$  is accepting if and only if  $h(s')$  accepting by definition of homomorphism; then  $\langle s \rangle_S$  contains an accepting state if and only if  $\langle h(s) \rangle_U$  contains an accepting state. Therefore,  $s$  is live in  $S$  if and only if  $h(s)$  is live in  $U$ .  $\square$

The above theorem then leads to several interesting liveness preservation properties for structures on coalgebras, like sub-coalgebra and bisimulation.

**Corollary 7** (sub-coalgebra preserves liveness). *For a sub-coalgebra  $S' \sqsubseteq S$  and a state  $s \in S'$ ,  $s$  is live in  $S'$  if and only if  $s$  is live in  $S$ .*

*Proof.* Let the homomorphism  $h$  in theorem 6 be the inclusion homomorphism  $i : S' \rightarrow S$ .  $\square$

**Corollary 8** (bisimulation preserves liveness). *If there exists a bisimulation  $\sim$  between GKAT coalgebra  $S$  and  $U$  s.t.  $s \sim u$  for some states  $s \in S$  and  $u \in U$ , then  $s$  and  $u$  has to be either both accepting, both live or both dead.*

*Proof.* Because for a  $\sim$  is a bisimulation when both  $\pi_1 : \sim \rightarrow S$  and  $\pi_2 : \sim \rightarrow U$  are homomorphisms. Therefore,

$$\begin{aligned} s \text{ is live in } S &\iff \pi_1((s, u)) \text{ is live in } S \\ &\iff (s, u) \text{ is live in } \sim \\ &\iff \pi_2((s, u)) \text{ is live in } U \\ &\iff u \text{ is live in } U. \end{aligned} \quad \square$$

#### D. Normalization And Semantics

Infinite-trace model  $\mathcal{G}_\omega$  is the final coalgebra of GKAT coalgebras [7]. The finality of the model allows us to define the semantics of each state in a given GKAT coalgebra  $S$ , which we will denote as  $\llbracket - \rrbracket_S^\omega : S \rightarrow \mathcal{G}_\omega$ , where the semantic equivalences can identified by bisimulation [7]:  $\llbracket s \rrbracket_S^\omega = \llbracket t \rrbracket_T^\omega$  if and only if there exists a bisimulation  $\sim \subseteq S \times T$ , s.t.  $s \sim t$ .

The infinite trace equivalences can be directly computed with bisimulation on derivative, which supports on-the-fly algorithm as demonstrated by similar systems [3, 1, 5]. However, the *finite* trace model  $\mathcal{G}$  is the final coalgebra of GKAT coalgebras without dead states, which we call *normal GKAT coalgebra* [8]. Fortunately every GKAT coalgebra can be normalized by rerouting all the transition from dead states to rejection. Concretely, given a GKAT

coalgebra  $S \triangleq (S, \delta_S)$ ,  $\delta_{\text{norm}(S)} : S \rightarrow G(S)$  is defined as  $\delta_{\text{norm}(S)}(s, \alpha) \triangleq \text{rej}$  when  $\delta_S(s, \alpha) = (s', p)$  and  $s'$  is dead in  $S$ ; and  $\delta_{\text{norm}(S)}(s, \alpha) \triangleq \delta_S(s, \alpha)$  otherwise. We call  $\text{norm}(S) \triangleq (S, \delta_{\text{norm}(S)})$  the *normalized* coalgebra of  $S$ .

And we use  $\llbracket - \rrbracket_S : \text{norm}(S) \rightarrow \mathcal{G}$  to denote the finite trace semantics of GKAT coalgebra, which is the unique homomorphism into the final coalgebra  $\mathcal{G}$ . The finite trace equivalence between  $s \in S$  and  $u \in U$  can be decided by first normalizing  $S$  and  $U$  then deciding whether there is a bisimulation  $\sim \subseteq \text{norm}(S) \times \text{norm}(U)$  s.t.  $s \sim t$ . For a more detailed explanation on the finite trace semantics, we refer the reader to the work of Smolka et al. [8], however we will recall the correctness theorem here.

**Theorem 9** (Correctness [8]). *Given two states in two GKAT coalgebra  $s \in S$  and  $u \in U$ , then there exists a bisimulation between normalized coalgebras  $\sim \subseteq \text{norm}(S) \times \text{norm}(U)$  s.t.  $s \sim u$  if and only if  $s$  and  $u$  are trace equivalent  $\llbracket s \rrbracket_S = \llbracket u \rrbracket_U$*

Besides giving us the finite-trace semantics, the normalization operation also connects the finite and infinite trace semantics, because it is an endofunctor on the category of GKAT coalgebra.

**Theorem 10.** *norm is an endofunctor in the category GKAT coalgebra. More specifically, if  $h : S \rightarrow U$  is a GKAT homomorphism, then  $h : \text{norm}(S) \rightarrow \text{norm}(U)$  is also a homomorphism.*

*Proof.* Recall that  $h$  is a homomorphism if and only if for all  $s \in S$  and  $\alpha \in \mathbf{At}$ :

- for a result  $r \in \{\text{rej}, \text{acc}\}$ ,

$$\delta_S(s, \alpha) = r \iff \delta_U(h(s), \alpha) = r;$$

- for any  $s' \in S$  and  $p \in K$ ,

$$\delta_S(s, \alpha) = (s', p) \iff \delta_U(h(s), \alpha) = (h(s'), p).$$

Then we show that  $h : \text{norm}(S) \rightarrow \text{norm}(U)$  is a homomorphism, this is a consequence of homomorphism preserves liveness (Theorem 6): for all  $s \in \text{norm}(S)$  and  $\alpha \in \mathbf{At}$ :

$$\begin{aligned} &\delta_{\text{norm}(S)}(s, \alpha) = \text{acc} \\ &\iff \delta_S(s, \alpha) = \text{acc} \\ &\iff \delta_U(h(s), \alpha) = \text{acc} \\ &\iff \delta_{\text{norm}(U)}(h(s), \alpha) = \text{acc}; \\ &\delta_{\text{norm}(S)}(s, \alpha) = \text{rej} \\ &\iff \delta_S(s, \alpha) = \text{rej} \text{ or } \delta_S(s, \alpha) = (s', p), s' \text{ is dead} \\ &\iff \delta_U(h(s), \alpha) = \text{rej} \\ &\quad \text{or } \delta_U(h(s), \alpha) = (h(s'), p), h(s') \text{ is dead} \\ &\iff \delta_{\text{norm}(U)}(h(s), \alpha) = \text{rej}; \end{aligned}$$

$$\begin{aligned}
& \delta_{\text{norm}(S)}(s, \alpha) = (s', p) \\
& \iff \delta_S(s, \alpha) = (s', p), s' \text{ is live} \\
& \iff \delta_U(h(s), \alpha) = (h(s'), p), h(s') \text{ is live} \\
& \iff \delta_{\text{norm}(U)}(h(s), \alpha) = (h(s'), p). \quad \square
\end{aligned}$$

**Corollary 11.** *Normalization preserves sub-coalgebra, i.e. if  $S' \sqsubseteq S$  then  $\text{norm}(S') \sqsubseteq \text{norm}(S)$ .*

*Proof.* By letting the homomorphism in Theorem 10 to be the inclusion homomorphism  $i : S' \rightarrow S$   $\square$

Because of the functoriality, we can show that two states are infinite-trace equivalent implies these two states are finite-trace equivalent. This gives us more tool in proving semantic equivalence between two states in GKAT coalgebras: proving bisimulation of these two states in the *non-normalized* GKAT coalgebra can also obtain semantic equivalence for two states.

**Corollary 12.** *Given two states in two GKAT coalgebra  $s \in S, u \in U$ ,  $\llbracket s \rrbracket_S^\omega = \llbracket u \rrbracket_U^\omega \implies \llbracket s \rrbracket_S = \llbracket u \rrbracket_U$ .*

*Proof.* Because  $\llbracket s \rrbracket_S^\omega = \llbracket u \rrbracket_U^\omega$ , there exists a bisimulation  $\sim \subseteq S \times U$  s.t.  $s \sim u$  [7]. Therefore, we have the following span in the category of GKAT coalgebra:

$$S \xleftarrow{\pi_1} \sim \xrightarrow{\pi_2} U$$

Then by Theorem 10,  $\text{norm}(\sim)$  is a bisimulation between  $\text{norm}(S)$  and  $\text{norm}(U)$ :

$$\text{norm}(S) \xleftarrow{\pi_1} \text{norm}(\sim) \xrightarrow{\pi_2} \text{norm}(U)$$

Because  $s \sim u$  and normalization operation preserves states in  $\sim$ , therefore  $(s, u) \in \text{norm}(\sim)$ , and because  $\text{norm}(\sim)$  is a bisimulation between the normalization of  $S$  and  $U$ , therefore  $\llbracket s \rrbracket_S = \llbracket u \rrbracket_U$  (Theorem 9).  $\square$

### III. ON-THE-FLY BISIMULATION

The original algorithm for deciding GKAT equivalences [8] requires the entire coalgebra to be known prior to the execution of the bisimulation algorithm; specifically, in order to compute the liveness of a state  $s$ , it is necessary iterate through all its reachable states  $\langle s \rangle$  to see if there are any accepting states within. This limitation poses challenges to design an efficient on-the-fly algorithm for GKAT. In order to make the decision procedure scalable, we will need to merge the normalization and bisimulation procedure, so that our algorithm can normalize the coalgebra only when we need to.

In this section, we introduce an algorithm that merges bisimulation and normalization, where we only test the liveness of states when a disparity is noticed by the bisimulation algorithm. For example, when deciding the trace equivalence between  $s \in S$  and  $u \in U$ , if  $\delta_S(s, \alpha) = (s', p)$  and  $\delta_U(u, \alpha) = (u', p)$ ,

we will proceed to recurse on  $s'$  and  $u'$ , without checking the liveness of  $s'$  and  $u'$ . However, if  $s$  rejects  $\alpha$  instead of transitioning to  $s'$ , we will then check whether  $u'$  is dead.

We present several sound and complete conditions for finite-trace equivalence between states of GKAT coalgebras in Theorem 9, which also serves as the core correctness theorem for our equivalence-checking algorithm i.e. Algorithm 1. This algorithm mostly consists of recursively checking the conditions in Theorem 9 with minor optimizations, which we will outline later.

**Lemma 13** (bisimulation between dead states). *Given two dead states  $s \in S$  and  $u \in U$ , then the singleton bisimulation  $\sim \subseteq \text{norm}(S) \times \text{norm}(U)$ :*

$$\sim \triangleq \{(s, u)\} \quad \delta_\sim((s, u), \alpha) \triangleq \text{rej}$$

*is a bisimulation between  $\text{norm}(S)$  and  $\text{norm}(U)$ .*

*Proof.* By computation  $\square$

**Lemma 14.** *Given two GKAT coalgebra  $S$  and  $U$ , and two of their elements  $s \in S$  and  $u \in U$ , there exists a bisimulation  $\sim \subseteq \text{norm}(\langle s \rangle) \times \text{norm}(\langle u \rangle)$  s.t.  $s \sim u$ , if and only if all the following holds:*

- 1) for all  $\alpha \in \mathbf{At}$ ,  $\delta_S(s, \alpha) = \text{acc} \iff \delta_U(u, \alpha) = \text{acc}$ ;
- 2)  $s$  reject  $\alpha$  or transition to a dead state via  $\alpha$  if and only if  $u$  rejects  $\alpha$  or transition to a dead state via  $\alpha$ ;
- 3) If  $\delta_S(s, \alpha) = (s', p)$  and  $\delta_U(u, \alpha) = (u', p)$ , then there exists a bisimulation  $\sim' \subseteq \text{norm}(\langle s' \rangle) \times \text{norm}(\langle u' \rangle)$ , s.t.  $s' \sim' u'$ ;
- 4) If  $\delta_S(s, \alpha) = (s', p)$  and  $\delta_U(u, \alpha) = (u', q)$ , s.t.  $p \neq q$ , then both  $s'$  and  $u'$  are dead.

*Proof.* We first prove  $\implies$  direction, recall that there exists a bisimulation  $\sim \subseteq \text{norm}(\langle s \rangle) \times \text{norm}(\langle u \rangle)$  if and only if for all  $s_1 \sim u_1$ :

- for all results  $r \in \{\text{acc}, \text{rej}\}$ :  $\delta_{\text{norm}(S)}(s_1, \alpha) = r \iff \delta_{\text{norm}(U)}(u_1, \alpha) = r$ ;
- otherwise, let  $(s_2, p) \triangleq \delta_{\text{norm}(S)}(s_1, \alpha)$  and  $(u_2, q) \triangleq \delta_{\text{norm}(U)}(u_1, \alpha)$ , then  $p = q$  and  $s_2 \sim u_2$

The condition 1 holds:

$$\begin{aligned}
\delta_S(s, \alpha) = \text{acc} & \iff \delta_{\text{norm}(S)}(s, \alpha) = \text{acc} \\
& \iff \delta_\sim((s, u), \alpha) = \text{acc} \\
& \iff \delta_{\text{norm}(U)}(u, \alpha) = \text{acc} \\
& \iff \delta_U(u, \alpha) = \text{acc}
\end{aligned}$$

The condition 2 holds:

$$\begin{aligned}
& \delta_S(s, \alpha) \text{ rejects or transition to dead states} \\
& \iff \delta_{\text{norm}(S)}(s, \alpha) = \text{rej} \\
& \iff \delta_{\text{norm}(U)}(u, \alpha) = \text{rej}
\end{aligned}$$

$\iff \delta_U(u, \alpha)$  rejects or transition to dead states.

The condition 3 holds, by case analysis on the liveness of  $s'$  and  $u'$ . First note that  $s'$  and  $u'$  has to be both live or both dead: because  $\delta_S(s, \alpha) = (s', p)$ , then  $\text{norm}(\delta_S)(s', \alpha)$  can either be rejection or  $(s', p)$ , and so is  $\text{norm}(\delta_U)(u', \alpha)$ :

$$\begin{aligned} s' \text{ is live} &\iff \delta_{\text{norm}(S)}(s, \alpha) = (s', p) \\ &\iff \delta_{\text{norm}(U)}(u, \alpha) = (u', p) \\ &\iff u' \text{ is live.} \end{aligned}$$

- If both  $s'$  and  $u'$  are live, then  $s' \sim u'$ . By theorem 4, the bisimulation  $\sim'$  is just  $\sim$  restricted to  $\langle s' \rangle$  and  $\langle u' \rangle$ .
- If both  $s'$  and  $u'$  are dead, then  $\sim'$  can just be the singleton relation, according to lemma 13.

The condition 4 holds: by the proof of condition 3,  $s'$  and  $u'$  has to be either both live or both dead; if they are both live, then there cannot be a element in  $G(\sim)$  that can project to  $(s', p)$  under  $\pi_1$  but projects to  $(t', q)$  under  $\pi_2$ . Thus both  $s'$  and  $t'$  has to be dead.

We then show the  $\Leftarrow$  direction, for arbitrary  $s' \in S$  and  $u' \in U$ , we use  $\equiv_{s', u'}$  to denote the maximal bisimulation between  $\text{norm}(\langle s' \rangle)$  and  $\text{norm}(\langle u' \rangle)$ .

$$\begin{aligned} \sim' &\triangleq \bigcup \{ \equiv_{s', u'} \mid \exists \alpha \in \mathbf{At}, p \in K, \\ &\delta_{\text{norm}(S)}(s, \alpha) = (s', p) \\ &\text{and } \delta_{\text{norm}(U)}(u, \alpha) = (u', p) \}. \end{aligned}$$

For all the  $s'$  and  $u'$  in the above definition,  $\langle s' \rangle \sqsubseteq \langle s \rangle$  and  $\langle u' \rangle \sqsubseteq \langle u \rangle$ , therefore by Corollary 11,  $\text{norm}(\langle s' \rangle) \sqsubseteq \text{norm}(\langle s \rangle)$  and  $\text{norm}(\langle u' \rangle) \sqsubseteq \text{norm}(\langle u \rangle)$ . By Theorem 4, every  $\equiv_{s', u'}$  is a bisimulation between  $\text{norm}(\langle s \rangle)$  and  $\text{norm}(\langle t \rangle)$ , and because bisimulation is closed under arbitrary union [6],  $\sim'$  is a bisimulation between  $\text{norm}(\langle s \rangle)$  and  $\text{norm}(\langle t \rangle)$ .

To obtain the desired bisimulation  $\sim$  between  $\text{norm}(\langle s \rangle)$  and  $\text{norm}(\langle u \rangle)$ , we add the pair  $(s, t)$  to  $\sim'$ ,

$$\sim \triangleq \sim' \cup \{(s, u)\},$$

with the following transition  $\delta_\sim$ : for all  $\alpha \in \mathbf{At}$ ,

- if  $\delta_{\text{norm}(S)}(s, \alpha) = \delta_{\text{norm}(U)}(u, \alpha) = \text{acc}$ , then  $\delta_\sim((s, u), \alpha) \triangleq \text{acc}$ ;
- if  $\delta_{\text{norm}(S)}(s, \alpha) = \delta_{\text{norm}(U)}(u, \alpha) = \text{rej}$ , then  $\delta_\sim((s, u), \alpha) \triangleq \text{rej}$ ;
- if  $\delta_{\text{norm}(S)}(s, \alpha) = (s', p)$  and  $\delta_{\text{norm}(U)}(u, \alpha) = (u', p)$ , then  $\delta_\sim((s, u), \alpha) = ((s', u'), p)$ ;
- for all  $(s', u') \in \sim'$  that is not equal to  $(s, u)$ , we let  $\delta_\sim$  inherits the transition of  $\delta_{\sim'}$ , i.e.  $\delta_\sim((s', u'), \alpha) = \delta_{\sim'}((s', u'), \alpha)$

if  $\delta_\sim$  is well-defined, then we can verify that  $\sim$  is indeed a bisimulation between  $\text{norm}(\langle s \rangle)$  and  $\text{norm}(\langle u \rangle)$  where  $s \sim u$ . We show the slightly more complicated case:  $\delta_{\text{norm}(S)}(s, \alpha) = (s', p)$  and

$\delta_{\text{norm}(U)}(u, \alpha) = (u', p)$  implies  $s' \sim u'$  as an example. By condition 3, there exists a bisimulation  $\sim_{s', u'} \subseteq \text{norm}(\langle s' \rangle) \times \text{norm}(\langle u' \rangle)$ , and because  $\equiv_{s', u'}$  is the maximal bisimulation between  $\text{norm}(\langle s' \rangle)$  and  $\text{norm}(\langle u' \rangle)$ ,

$$(s', u') \in \sim_{s', u'} \subseteq \equiv_{s', u'} \subseteq \sim.$$

Finally, we demonstrate that  $\delta_\sim$  is well-defined by leveraging the conditions in Theorem 15. Specifically, we will show that the definition of  $\delta_\sim$  covers all the possible cases, by case analysis on the result of  $\delta_S$ : for all  $\alpha \in \mathbf{At}$ ,

- If  $\delta_S(s, \alpha) = \text{acc}$ , then by condition 1,  $\delta_U(u, \alpha) = \text{acc}$ ; therefore

$$\delta_{\text{norm}(S)}(s, \alpha) = \delta_{\text{norm}(U)}(u, \alpha) = \text{acc}.$$

- If  $\delta_S(s, \alpha)$  transitions to a dead state or reject, then by condition 2  $\delta_U(u, \alpha)$  will also transition to a dead state or reject, then

$$\delta_{\text{norm}(S)}(s, \alpha) = \delta_{\text{norm}(U)}(u, \alpha) = \text{rej}.$$

- If  $\delta_S(s, \alpha) = (s', p)$  and  $s'$  is live, then  $\delta_U(u, \alpha) = (u', p)$  necessarily holds, otherwise it would violate one of conditions 1, 2 and 4.

By condition 3, there exists a bisimulation  $\sim_{s', u'}$  between  $\text{norm}(\langle s' \rangle)$  and  $\text{norm}(\langle u' \rangle)$  s.t.  $s' \sim_{s', u'} u'$ . Because bisimulation preserves liveness (Corollary 8),  $s', u'$  has to be both dead or live. Finally, because  $s'$  is live, therefore  $u'$  is also live, and we obtain the final case in the definition of  $\delta_\sim$ :  $\delta_{\text{norm}(S)}(s, \alpha) = (s', p)$  and  $\delta_{\text{norm}(U)}(u, \alpha) = (u', p)$ .  $\square$

With some slight tweak to Lemma 14, it will be applicable to trace equivalence instead of bisimulation. The correctness of this modification is supported by the correspondence between trace equivalence and the existence of a bisimulation between normalized GKAT coalgebras, which is stated in Theorem 9.

**Theorem 15** (Recursive Construction). *For any two states in two GKAT coalgebra  $s \in S, u \in U$ ,  $s$  and  $u$  are finite-trace equivalent  $\llbracket s \rrbracket_S = \llbracket u \rrbracket_U$  if and only if all the following conditions hold:*

- 1) for all  $\alpha \in \mathbf{At}$ ,  $\delta_S(s, \alpha) = \text{acc} \iff \delta_U(u, \alpha) = \text{acc}$ ;
- 2)  $s$  reject  $\alpha$  or transition to a dead state via  $\alpha$  if and only if  $u$  rejects  $\alpha$  or transition to a dead state via  $\alpha$ ;
- 3) if  $\delta_S(s, \alpha) = (s', p)$  and  $\delta_U(u, \alpha) = (u', p)$ , then  $\llbracket s \rrbracket_S = \llbracket u \rrbracket_U$ ;
- 4) if  $\delta_S(s, \alpha) = (s', p)$  and  $\delta_U(u, \alpha) = (u', q)$ , s.t.  $p \neq q$ , then both  $s'$  and  $t'$  are dead.

Notice that above condition is similar to those in theorem 15, except bisimulation of  $s'$  and  $u'$  is replaced with trace equivalence.

*Proof.* By the standard argument with normalization preserves sub-coalgebra (Corollary 11), we can obtain for all  $s \in S$  and  $u \in U$ ,  $\text{norm}(\langle s \rangle) \sqsubseteq \text{norm}(S)$  and  $\text{norm}(\langle u \rangle) \sqsubseteq \text{norm}(U)$ . Therefore, because subcoalgebra preserve and reflect bisimulation (Theorem 4) and the correctness of bisimulation on normalized coalgebra (Theorem 9): for all  $s \in S$  and  $u \in U$ ,

$$\begin{aligned} \exists \sim &\subseteq \text{norm}(\langle s \rangle) \times \text{norm}(\langle u \rangle), s \sim u \\ \iff \exists \sim &\subseteq \text{norm}(S) \times \text{norm}(U), s \sim u \\ \iff \llbracket s \rrbracket_S &= \llbracket u \rrbracket_U. \end{aligned}$$

We can instantiate the above equivalence to  $s', u'$  and  $s, u$  respectively: the instantiation to  $s', u'$  will give us the conditions in this theorem is equivalent to the ones in Lemma 14; and instantiation to  $s, u$  show that existence of bisimulation is equivalent to trace equivalence:

$$\begin{aligned} &\text{Conditions in Lemma 14 hold} \\ \iff &\text{Conditions in the current theorem hold} \\ \iff \exists \sim &\subseteq \text{norm}(\langle s \rangle) \times \text{norm}(\langle u \rangle), s \sim u \\ \iff \llbracket s \rrbracket_S &= \llbracket u \rrbracket_U. \quad \square \end{aligned}$$

The above construction theorem already gives us an algorithm to recursively decide whether  $\llbracket s \rrbracket_S = \llbracket u \rrbracket_U$ , when  $\langle s \rangle_S$  and  $\langle u \rangle_U$  is finite. However, this algorithm can be further optimized: we will derive that a dead state is only trace equivalent to other dead states. This means that when checking the trace-equivalence of states  $s$  and  $u$ , if we already know one of them is dead, we only need to check whether the other is dead, instead of going through all the conditions in Theorem 15.

**Theorem 16.** *Given two states  $s \in S$  and  $u \in U$ , if  $s$  is a dead state in  $S$ , then  $s$  and  $u$  is trace equivalent if and only if  $u$  is also dead.*

*Proof.* TODO: I think we can refine  $\implies$  direction, but I am not sure...

For the  $\impliedby$  direction, we can construct the bisimulation as in lemma 13, which implies trace equivalence. And the  $\implies$  direction, if  $s$  is dead, then by definition of normalization, it will be all rejecting, i.e. for all  $\alpha \in \mathbf{At}$ ,  $\delta_{\text{norm}(S)}(s, \alpha) = \text{rej}$ . If  $s$  and  $u$  are trace equivalent, then there exists a bisimulation  $\sim : \text{norm}(S) \times \text{norm}(U)$  s.t.  $s \sim u$ . By unfolding the definition of a bisimulation,  $u$  also need to be all rejecting in  $\text{norm}(U)$ . By definition of norm, for all  $\alpha \in \mathbf{At}$ ,  $\delta_U(u, \alpha)$  either reject to go to a dead state,

and because  $\langle u \rangle_U$  is all the reachable state from  $u$ , therefore

$$\langle u \rangle_U = \bigcup \{ \langle u' \rangle \mid \exists \alpha \in \mathbf{At}, p \in K, \delta_U(u, \alpha) = (u', p) \} \cup \{u\}$$

And because all of  $u'$  is dead, all of  $\langle u' \rangle_U$  cannot be accepting (Lemma 5), nor is  $u$  an accepting state, then  $\langle u \rangle_U$  does not contain any accepting state, and by definition of dead state,  $u$  is dead.  $\square$

Before we introduce the optimized algorithm, we will first briefly sketch the liveness-check algorithm, which will use a depth-first search to check whether there exists any accepting states in reachable states of  $s \in S$ : if there exists any accepting state in  $\langle s \rangle_S$ , then the algorithm terminates immediately, and return that  $s$  is live, otherwise it would return all the states in  $\langle s \rangle$ , and by lemma 5, all the states in  $\langle s \rangle$  is dead. In this case, we use depth-first search for its simplicity to implement, other search algorithm will also be sound.

We will cache all the known dead states from the previous searches; whether a state  $s$  is in this cache can be checked by the function call  $\text{KNOWNDEAD}_S(s)$ . The function  $\text{ISDEAD}_S(s)$  will first check if  $s$  is known to be dead, and invoke the search algorithm in the coalgebra  $\langle s \rangle$ , if  $s$  is not in the cached dead states. Because the non-symbolic version of liveness checking is similar to the symbolic version, we only provide the symbolic version of this algorithm in appendix TODO: give the link to the section.

Finally, we present our equivalence-checking algorithm as algorithm 1, where we first check if one of  $s$  or  $u$  is known to be dead, if so we only need to check whether the other is dead, because of Theorem 16; otherwise, we will check the conditions in Theorem 15. Because trace equivalence is an equivalence relation, therefore we can organize the previously explored states into equivalent classes using an efficient union-find structure, instead of a set of state pairs. This union-find structure need to provide two functions:  $\text{UNION}(s, u)$  unions the equivalence class of  $s$  and  $u$ , whereas  $\text{EQ}(s, u)$  checks whether  $s$  and  $u$  is in the same equivalence class.

Concerning the complexity, our algorithm have similar worst case complexity as the original algorithm. In particular, when deciding the trace equivalence of two states  $s \in S$  and  $u \in U$ , the original algorithm [8] requires one pass of  $\langle s \rangle$  and  $\langle u \rangle$  to normalize them, then the bisimulation will visit each pair of states in  $\langle s \rangle \times \langle u \rangle$  at most once, which means that they will visit at most  $|\langle s \rangle| + |\langle u \rangle| + |\langle s \rangle \times \langle u \rangle|$  number of states.

Similarly, our equivalence-checking algorithm attempts to find a bisimulation first, which visits each

---

**Algorithm 1** On-the-fly bisimulation algorithm

---

```

function EQUIV( $s \in S, u \in U$ )
  if EQ( $s, u$ ) then return true
  if KNOWNDEAD $_S(s)$  then
    return ISDEAD $_U(u)$ 
  if KNOWNDEAD $_U(u)$  then
    return ISDEAD $_S(s)$ 
  for  $\alpha \in \mathbf{At}$  do
    match  $\delta_S(s, \alpha), \delta_U(u, \alpha)$  with
      case acc, acc then continue
      case rej, rej then continue
      case rej, ( $u', q$ ) then
        return ISDEAD $_U(u')$ 
      case ( $s', p$ ), rej then
        return ISDEAD $_S(s')$ 
      case ( $s', p$ ), ( $u', q$ ) then
        if  $p = q$  then
          UNION( $s, u$ )
          return EQUIV( $s, t$ )
        if ISDEAD $_S(s)$  and ISDEAD $_U(u)$  then
          continue
        return false
      default return false
  return true

```

---

pair in  $\langle s \rangle \times \langle u \rangle$  at most once, and when a mismatch is found in the process, the liveness-checking algorithm is then invoked. Crucially, our algorithm satisfy the following property: if the liveness-checking algorithm find a live or accepting state, the entire equivalence checking algorithm will halt and return false. Thus, by caching all the known dead states, the liveness-checking only visits states in  $\langle s \rangle$  and  $\langle u \rangle$  at most once. Therefore, the worst case of our algorithm is also  $|\langle s \rangle \times \langle u \rangle| + |\langle s \rangle| + |\langle u \rangle|$ .

However, the on-the-fly algorithm will always outperform the original algorithm, in terms of number of states visited, because this algorithm only invoke liveness check when necessary. In the extreme case when the two input states are infinite-trace equivalent, the on-the-fly algorithm can skip liveness checking entirely.

#### IV. SYMBOLIC COALGEBRA AND ALGORITHM

Although algorithm 1 is on-the-fly, it still uses GKAT coalgebra, which contains exponentially many transitions with respect to the number of primitive tests  $|T|$ . Concretely, the transition function  $\delta : S \rightarrow \mathbf{At}_T \rightarrow \{\text{acc}, \text{rej}\} + S \times K$  requires computing the transition result for each *atom*, and the number of atoms  $\mathbf{At}_T \cong 2^T$  is exponential to the size of primitive

tests in  $T$ . This is also why several GKAT-related complexity results only consider constant sized  $T$ .

Symbolic GKAT coalgebra, instead of computing the behavior of each atom individually, groups atoms into boolean expressions. This optimization leads to space-efficient coalgebras and an equivalence checking algorithm making use of off-the-shelf SAT solvers. Specifically, given a set of primitive actions  $K$  and primitive tests  $T$ , a *symbolic GKAT coalgebra*  $\hat{S} \triangleq (S, \hat{\epsilon}, \hat{\delta})$  consists of a state set  $S$  and an accepting function  $\hat{\epsilon}$  and a transition function  $\hat{\delta}$ :

$$\hat{\epsilon} : S \rightarrow \mathcal{P}(\text{BExp}_T), \quad \hat{\delta} : S \rightarrow \mathcal{P}(\text{BExp}_T \times S \times K).$$

This coalgebra is also required to satisfy the disjointedness condition, i.e. for all states  $s \in S$ , the conjunction of any two expressions from the set  $\hat{\epsilon}(s) \cup \{b \mid \exists(b, s', p) \in \hat{\delta}(s)\}$  is equivalent to 0 under boolean algebra.

We name  $\hat{\epsilon}$  the accepting function because intuitively a state  $s$  accepts an atom  $\alpha$  when there exists a  $b \in \hat{\epsilon}(s)$ , s.t.  $\alpha \leq b$ ; similarly,  $\hat{\delta}$  is called the transition function because a state  $s$  transitions to  $s'$  via atom  $\alpha$  while executing  $p$  when there exists  $(b, s', p) \in \hat{\delta}(s)$  and  $\alpha \leq b$ . With the above intuition in mind, a symbolic GKAT coalgebra  $\hat{S} \triangleq (S, \hat{\epsilon}, \hat{\delta})$  can be lowered into a GKAT coalgebra  $S \triangleq (S, \delta)$  in the following manner:

$$\delta(s, \alpha) \triangleq \begin{cases} \text{acc} & \exists b \in \hat{\epsilon}(s), \alpha \leq b \\ (s', p) & \exists b \in \text{BExp}_T, \alpha \leq b \\ & \text{and } \delta(s, b) = (s', p) \\ \text{rej} & \text{otherwise} \end{cases} \quad (1)$$

This is well-defined, i.e. exactly one clause can be satisfied for any  $s \in S$  and  $\alpha \in \mathbf{At}$ , because of the disjointedness condition. We usually use  $S$  to denote the lowering of  $\hat{S}$ ; and the semantics of a state  $s \in \hat{S}$  is defined as its semantics in the lowering  $\llbracket s \rrbracket_{\hat{S}} \triangleq \llbracket s \rrbracket_S$ .

We will then use  $\hat{\rho}(s) : \text{BExp}_T$  to represent all the atoms that the state  $s$  rejects in the lowering, and  $\hat{\rho}(s)$  is computed as follows:

$$\hat{\rho}(s) \triangleq \bigwedge \{ \bar{b} \mid \exists s' \in S, p \in K, (b, s', p) \in \hat{\delta}(s) \\ \text{or } b \in \hat{\epsilon}(s) \}.$$

**Remark 2** (Canonicity). *Symbolic GKAT coalgebra is not canonical, i.e. there exists two different symbolic GKAT coalgebra with the same lowering, consider the state set  $S \triangleq \{s\}$ :*

$$\begin{aligned} \hat{\delta}_1(s) &\triangleq \{b \mapsto (s, p), \bar{b} \mapsto (s, p)\} \\ \hat{\delta}_2(s) &\triangleq \{\top \mapsto (s, p)\}, \end{aligned}$$

and both  $\hat{\epsilon}_1, \hat{\epsilon}_2$  will return constant 0. These two symbolic GKAT coalgebra  $\hat{S}_1 \triangleq (S, \hat{\delta}_1, \hat{\epsilon}_1)$  and  $\hat{S}_2 \triangleq$



$(S, \hat{\delta}_2, \hat{\epsilon}_2)$  have the same lowering and semantics, yet, they are different. It is possible to construct symbolic representations that satisfies canonicity, yet we opt to use our current representation for ease of construction and computational efficiency.

**Theorem 17** (Functoriality). *The lowering operation is a functor, every symbolic GKAT coalgebra homomorphism  $h : \hat{S} \rightarrow \hat{U}$ , is also a GKAT coalgebra homomorphism  $h : S \rightarrow U$ .*

*Proof.* Since  $\hat{S}$  and  $\hat{U}$  have the same states as their lowering, therefore  $h : S \rightarrow U$  is indeed a function, then we only need to verify the homomorphism condition on  $h$ . TODO: finish.  $\square$

The functoriality states that a homomorphism on two symbolic coalgebras induces a homomorphism on their lowering; similarly, a bisimulation on symbolic GKAT coalgebra also induces a bisimulation on their lowering. However, the converse is not true, precisely because of the canonicity problem noted in Remark 2: take the  $\hat{S}_1$  and  $\hat{S}_2$  in Remark 2, because they have the same lowering, therefore the identity homomorphism is a homomorphism on their lowerings, but there is no homomorphism from  $\hat{S}_1$  to  $\hat{S}_2$ .

We can then define the symbolic equivalence algorithm, with its correctness stated in theorem 15.

**Theorem 18** (Symbolic Recursive Construction). *Given two symbolic GKAT coalgebra  $\hat{S} = (S, \hat{\epsilon}_S, \hat{\delta}_S)$  and  $\hat{U} = (U, \hat{\epsilon}_U, \hat{\delta}_U)$  and two states  $s \in S$  and  $u \in U$ ,  $s$  and  $u$  are trace equivalent  $\llbracket s \rrbracket_{\hat{S}} = \llbracket u \rrbracket_{\hat{U}}$ , if and only if all the following holds:*

- $\bigvee \hat{\epsilon}_S(s) \equiv \bigvee \hat{\epsilon}_U(u)$ ;
- for all  $(b, s', p) \in \hat{\delta}_S(s)$  and  $c \in \hat{\rho}_U(u)$ , if  $b \wedge c \neq 0$ , then  $s'$  is dead;
- for all  $b \in \hat{\rho}_S(s)$  and  $(c, u', q) \in \hat{\delta}_U(u)$ , if  $b \wedge c \neq 0$ , then  $u'$  is dead;
- for all  $(b, s', p) \in \hat{\delta}_S(s)$  and  $(c, u', q) \in \hat{\delta}_U(u)$ , if  $b \wedge c \neq 0$  and  $p \neq q$  then both  $s'$  and  $u'$  is dead;
- for all  $(b, s', p) \in \hat{\delta}_S(s)$  and  $(c, u', q) \in \hat{\delta}_U(u)$ , if  $b \wedge c \neq 0$  and  $p = q$  then  $\llbracket s' \rrbracket_{\hat{S}} = \llbracket u' \rrbracket_{\hat{U}}$ .

*Proof.* Reduces to Theorem 15 i.e. all the above condition holds if and only if all the condition in Theorem 15 holds in the lowering.  $\square$

Similar to the non-symbolic version of the algorithm, we will first check if either of the input states is dead, then recursively check all the conditions in theorem 18. And the symbolic version of the liveness checking algorithm is also based on a graph search algorithm on all the reachable state, where we try to identify an accepting state by checking whether  $\bigvee \hat{\epsilon}(s) \equiv 0$ . The only deviation from the non-symbolic cases is that we do not follow empty transitions in the

symbolic case, i.e. if  $\hat{\delta}(s) \triangleq (b, s', p)$  and  $b \equiv 0$ , we will not search  $s'$  to check the liveness of  $s'$ . In Section VI-A, we will outline a simple modification for the symbolic GKAT coalgebra generation algorithm in Section V to avoid generating empty transitions, allowing us to skip the emptiness check when deciding whether a state is dead.

The symbolic trace-equivalence algorithm is presented as algorithm 2, where  $\&\&$  is the logical-and operator on the language level, specifying that all the conditions in theorem 18 must be satisfied for the algorithm to return true.

## V. SYMBOLIC GKAT COALGEBRA CONSTRUCTION

The final piece of the puzzle is to convert any given expression into an “equivalent” Symbolic GKAT Automata. This goal can be achieved by lifting existent constructions like derivatives and Thompson’s construction [7, 8]. The correctness of these conversions is a consequence of correctness of their non-symbolic counter-part, i.e. we will prove that the lowering as shown in (1) of these constructions will yield the conventional derivative and Thompson’s construction.

The symbolic derivative coalgebra  $\hat{D}$ , with expressions as states, is the least symbolic GKAT coalgebra (ordered by point-wise subset ordering on  $\hat{\epsilon}$  and  $\hat{\delta}$ ) that satisfy the rules in Figure 1. Notice that the rules listed on Figure 1 is very close to that of Schmid et al. [7]. This is no coincidence, as our definition exactly lowers to the definition of theirs. This fact can be proven by case analysis on the shape of the source expression, and forms a basis on our correctness argument.

**Theorem 19** (Correctness). *The lowering of  $\hat{D}$ , denoted  $D$ , is exactly the derivative defined by Schmid et al. [7]. Therefore, the semantics of the expression is equal to the semantics generated by the derivative coalgebra,  $\llbracket e \rrbracket = \llbracket e \rrbracket_D = \llbracket e \rrbracket_{\hat{D}}$ .*

*Proof.* TODO: unfold the statement.  $\square$

Another way to construct a coalgebra from a GKAT expression is via Thompson’s construction, we lift the original construction to the symbolic setting. A common expression to construct is a guard operation, denoted by  $\langle B |$ , where  $B$  is a set of boolean expressions. TODO: define transition dynamics and accepting dynamics earlier. Concretely, this guard can be defined on both accepting dynamics and transition dynamics:

$$\langle B | \hat{\epsilon}(s) \triangleq \{b \wedge c \mid b \in B, c \in \epsilon(s)\};$$

$$\langle B | \hat{\delta}(s) \triangleq \{(b \wedge c, s', p) \mid b \in B, (c, s', p) \in \delta(s)\}.$$

Notably, besides guarding transition and acceptance with different conditions in if statements and while

---

**Algorithm 2** Symbolic On-the-fly Bisimulation Algorithm

---

**function** EQUIV( $s \in S, u \in U$ )

**if** EQ( $s, u$ ) **then return** true

**else if** KNOWNDEAD $_S(s)$  **then return** ISDEAD $_U(u)$ 
**else if** KNOWNDEAD $_U(u)$  **then return** ISDEAD $_S(s)$ 
**else return**
 $\triangleright$  conditions of theorem 18

$$\bigvee \hat{e}_S(s) \equiv \bigvee \hat{e}_U(u) \ \&\&$$

$$\forall (b, s', p) \in \hat{\delta}_S(s), (c, u', q) \in \hat{\delta}_U(u), (b \wedge c) \neq 0 \implies \begin{cases} \text{ISDEAD}_S(s) \wedge \text{ISDEAD}_U(u) & \text{if } p \neq q \\ \text{UNION}(s, u); \text{EQUIV}(s', u') & \text{if } p = q \end{cases} \ \&\&$$

$$\forall (b, s', p) \in \hat{\delta}_S(s), c \in \hat{\rho}_U(u), (b \wedge c) \neq 0 \implies \text{ISDEAD}_S(s') \ \&\&$$

$$\forall b \in \hat{\rho}_S(s), (c, u', q) \in \hat{\delta}_U(u), (b \wedge c) \neq 0 \implies \text{ISDEAD}_U(u')$$


---

$$\begin{array}{c} \frac{}{p \xrightarrow{1p} 1} \quad \frac{}{b \Rightarrow_{\hat{D}} b} \quad \frac{e \xrightarrow{c|p} e'}{e +_b f \xrightarrow{b \wedge c|p} e'} \quad \frac{e \Rightarrow_{\hat{D}} c}{e +_b f \Rightarrow_{\hat{D}} b \wedge c} \quad \frac{f \xrightarrow{c|p} f'}{e +_b f \xrightarrow{\bar{b} \wedge c|p} f'} \quad \frac{f \Rightarrow_{\hat{D}} c}{e +_b f \Rightarrow_{\hat{D}} \bar{b} \wedge c} \\[10pt] \frac{e \Rightarrow_{\hat{D}} b \quad f \Rightarrow_{\hat{D}} c}{e; f \Rightarrow_{\hat{D}} b \wedge c} \quad \frac{e \xrightarrow{b|p} e'}{e; f \xrightarrow{b|p} e'; f} \quad \frac{e \Rightarrow_{\hat{D}} b \quad f \xrightarrow{c|p} f'}{e; f \xrightarrow{b \wedge c|p} f'} \quad \frac{}{e^{(b)} \Rightarrow_{\hat{D}} \bar{b}} \quad \frac{e \xrightarrow{c|p} e'}{e^{(b)} \xrightarrow{b \wedge c|p} e'; e^{(b)}} \end{array}$$

Fig. 1: Symbolic Derivative

Exp	$S$	$s^*$	$\hat{e}(s)$	$\hat{\delta}(s)$
$b$	$\{s^*\}$	$s^*$	$\{b\}$	$\emptyset$
$p$	$\{s^*, s_1\}$	$s^*$	$\begin{cases} \emptyset & s = s^* \\ \{1\} & s = s_1 \end{cases}$	$\begin{cases} \{(1, s_1, 0)\} & s = s^* \\ \emptyset & s = s_1 \end{cases}$
$e_1 +_b e_2$	$\{s^*\} + S_1 + S_2$	$s^*$	$\begin{cases} \langle \{b\}   \hat{e}_1(s_1^*) \cup \langle \{b\}   \hat{e}_2(s_2^*) & s = s^* \\ \hat{e}_1(s) & s \in S_1 \\ \hat{e}_2(s) & s \in S_2 \end{cases}$	$\begin{cases} \langle \{b\}   \hat{\delta}_1(s_1^*) + \langle \{b\}   \hat{\delta}_2(s_2^*) & s = s^* \\ \hat{\delta}_1(s) & s \in S_1 \\ \hat{\delta}_2(s) & s \in S_2 \end{cases}$
$e_1; e_2$	$S_1 + S_2$	$s_1^*$	$\begin{cases} \langle \hat{e}_1(s)   \hat{e}_2(s_2^*) & s \in S_1 \\ \hat{e}_2(s) & s \in S_2 \end{cases}$	$\begin{cases} \hat{\delta}_1(s) + \langle \hat{e}(s)   \hat{\delta}_2(s_2^*) & s \in S_1 \\ \hat{\delta}_2(s) & s \in S_2 \end{cases}$
$e_1^{(b)}$	$\{s^*\} + S_1$	$s^*$	$\begin{cases} \{\bar{b}\} & s = s^* \\ \langle \{\bar{b}\}   \hat{e}_1(s) & s \in S_1 \end{cases}$	$\begin{cases} \langle \{b\}   \hat{\delta}_1(s_1^*) & s = s^* \\ \hat{\delta}_1(s) \cup \langle \{b\}   \langle \hat{e}_1(s)   \hat{\delta}_1(s_1^*) & s \in S_1 \end{cases}$

TABLE I: Symbolic Thompson's Construction

loops, the guard operator can also be used to simulate uniform continuation. Specifically, we can use  $\langle \hat{e}(s) | \delta(s') \rangle$  to connecting all the accepting state of  $s$  to the dynamic  $\delta(s')$ .

With these definitions in mind, we can define symbolic Thompson's construction inductively as in Table I, where we let  $(S_1, \hat{e}_1, \hat{\delta}_1)$  and  $(S_2, \hat{e}_2, \hat{\delta}_2)$  to be result of Thompson's construction for  $e_1$  and  $e_2$  respectively. The  $S_1 + S_2$  is the disjoint union of  $S_1$  and  $S_2$ , and for any two transition dynamics  $\delta_1(s_1) : \mathcal{P}(\text{BExp} \times S_1 \times K)$  and  $\delta_2(s_2) : \mathcal{P}(\text{BExp} \times S_2 \times K)$ , then  $\delta_1(s_1) + \delta_2(s_2) : \mathcal{P}(\text{BExp} \times (S_1 + S_2) \times K)$  is defined as follows:

$$\begin{aligned} \delta_1(s_1) + \delta_2(s_2) &\triangleq \\ &\{(b, \text{inj}_l(s'_1), p) \mid (b, s'_1, p) \in \delta_1(s_1)\} \cup \end{aligned}$$

$$\{(b, \text{inj}_r(s'_2), p) \mid (b, s'_2, p) \in \delta_2(s_2)\}$$

where  $\text{inj}_l : S_1 \rightarrow S_1 + S_2$  and  $\text{inj}_r : S_2 \rightarrow S_1 + S_2$  are the canonical left/right injection of the coproduct.

One notable difference between the original construction [8] and our construction is that we use a start state  $s^* \in S$ , instead of a start dynamics (or pseudo-state). This choice will make the proof of Theorem 20 slightly easier. However, in Section VI-A, we will explain that our implementation uses start dynamics instead of start state, to avoid unnecessary lookups and unreachable states.

We would like to explore several desirable theoretical properties of both derivatives and Thompson's construction. Specifically, the *correctness*, i.e. the semantics of the “start state” the both construction

have the same preserves the trace semantics of the expression; *finiteness*, i.e. the coalgebra generated is always finite, which means that our equivalence algorithm will eventually terminate; and finally, *complexity*, the relationship between the number of reachable states and the size of the input expression, which serves as an estimated complexity of our equivalence checking algorithm. Turns out all of these questions can be answered by a homomorphism from symbolic Thompson's construction to the symbolic derivatives.

**Theorem 20.** *Given any GKAT expression  $e$ , the resulting symbolic GKAT coalgebra from Thompson's construction  $\hat{S}_e$  have a homomorphism to derivatives  $h : \hat{S}_e \rightarrow \hat{D}$ , s.t. for the start state  $s^* \in S$ ,  $h(s^*) = e$ .*

*Proof.* By induction on the structure of  $e$ . We will recall that  $h : \hat{S}_e \rightarrow \langle e \rangle_D$  is a symbolic GKAT coalgebra homomorphism when the following two conditions are true:  $s \Rightarrow_{S_e} b$  if and only if  $h(s) \Rightarrow_{\hat{D}} b$ ; and  $s \xrightarrow{b|p}_{S_e} s'$  if and only if  $h(s) \xrightarrow{b|p}_{\hat{D}} h(s')$ .

When  $e \triangleq b$  for some tests  $b$ , then the function  $h$  is defined as  $\{s^* \mapsto b\}$ . When  $e \triangleq p$  for some primitive action  $p$ , then the function  $h$  is defined as  $\{s^* \mapsto p, * \mapsto 1\}$ . The homomorphism condition can then be verified by unfolding the definition.

When  $e \triangleq e_1 +_b e_2$ , by induction hypothesis, we have homomorphisms  $h_1 : \hat{S}_{e_1} \rightarrow \langle e_1 \rangle_D$  and  $h_2 : \hat{S}_{e_2} \rightarrow \langle e_2 \rangle_D$ . Then we define the homomorphism

$$h(s) \triangleq \begin{cases} e_1 +_b e_2 & s = s^* \\ h_1(s) & s \in \hat{S}_{e_1} \\ h_2(s) & s \in \hat{S}_{e_2} \end{cases}$$

We show that  $h$  is a homomorphism. Because  $\hat{S}_e$  preserves the transition and acceptance of  $\hat{S}_{e_1}$  and  $\hat{S}_{e_2}$ , then for all  $s \in \hat{S}_{e_1} \cap \hat{S}_{e_2}$ , we have

$$\begin{aligned} s \Rightarrow_{\hat{S}_e} c &\text{ iff } s \Rightarrow_{\hat{S}_{e_1}} c \\ &\text{ iff } h_1(s) \Rightarrow_{\hat{D}} c \text{ iff } h(s) \Rightarrow_{\hat{D}} c; \\ s \xrightarrow{c|p}_{\hat{S}_e} s' &\text{ iff } s \xrightarrow{c|p}_{\hat{S}_{e_1}} s' \\ &\text{ iff } h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s') \text{ iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s'). \end{aligned}$$

And similarly for  $s \in \hat{S}_{e_2} \cap \hat{S}_e$ . So we only need to show the homomorphic condition for the start state  $s^*$ :

$$\begin{aligned} s^* \Rightarrow_{\hat{S}_e} c & \\ \text{iff } (\exists a, b \wedge a = c \text{ and } s_1^* \Rightarrow_{\hat{S}_{e_1}} a) & \\ \text{or } (\exists a, \bar{b} \wedge a = c \text{ and } s_2^* \Rightarrow_{\hat{S}_{e_2}} a) & \\ \text{iff } (\exists a, b \wedge a = c \text{ and } h_1(s_1^*) \Rightarrow_{\hat{D}} a) & \\ \text{or } (\exists a, \bar{b} \wedge a = c \text{ and } h_2(s_2^*) \Rightarrow_{\hat{D}} a) & \end{aligned}$$

$$\begin{aligned} &\text{iff } (\exists a, b \wedge a = c \text{ and } e_1 \Rightarrow_{\hat{D}} a) \\ &\text{or } (\exists a, \bar{b} \wedge a = c \text{ and } e_2 \Rightarrow_{\hat{D}} a) \\ \text{iff } e_1 +_b e_2 \Rightarrow_{\hat{D}} c & \\ \text{iff } h(s^*) \Rightarrow_{\hat{D}} c. & \\ s^* \xrightarrow{a|p}_{\hat{S}_e} s' & \\ \text{iff } (\exists a, b \wedge a = c \text{ and } s_1^* \xrightarrow{a|p}_{\hat{S}_{e_1}} s') & \\ \text{or } (\exists a, \bar{b} \wedge a = c \text{ and } s_2^* \xrightarrow{a|p}_{\hat{S}_{e_2}} s') & \\ \text{iff } (\exists a, b \wedge a = c \text{ and } h_1(s_1^*) \xrightarrow{a|p}_{\hat{D}} h(s')) & \\ \text{or } (\exists a, \bar{b} \wedge a = c \text{ and } h_2(s_2^*) \xrightarrow{a|p}_{\hat{D}} h(s')) & \\ \text{iff } (\exists a, b \wedge a = c \text{ and } e_1 \xrightarrow{a|p}_{\hat{D}} h(s')) & \\ \text{or } (\exists a, \bar{b} \wedge a = c \text{ and } e_2 \xrightarrow{a|p}_{\hat{D}} h(s')) & \\ \text{iff } e_1 +_b e_2 \xrightarrow{a|p}_{\hat{D}} h(s') & \\ \text{iff } h(s^*) \xrightarrow{a|p}_{\hat{D}} h(s'). & \end{aligned}$$

When  $e \triangleq e_1; e_2$ , by induction hypothesis, we have two homomorphisms  $h_1 : \hat{S}_{e_1} \rightarrow \hat{D}$  and  $h_2 : \hat{S}_{e_2} \rightarrow \hat{D}$ . We define  $h$  as follows:

$$h(s) \triangleq \begin{cases} h_1(s); e_2 & s \in \hat{S}_{e_1} \\ h_2(s) & s \in \hat{S}_{e_2} \end{cases}$$

Then we can prove that  $h$  is a homomorphism by case analysis on  $s$ . First case is that  $s \in \hat{S}_{e_1}$ :

$$\begin{aligned} s \Rightarrow_{\hat{S}_e} c & \\ \text{iff } \exists a, b, a \wedge b = c, s \Rightarrow_{\hat{S}_{e_1}} a \text{ and } s_2^* \Rightarrow_{\hat{S}_{e_2}} b & \\ \text{iff } \exists a, b, a \wedge b = c, h_1(s) \Rightarrow_{\hat{D}} a \text{ and } f \Rightarrow_{\hat{D}} b & \\ \text{iff } h_1(s); f \Rightarrow_{\hat{D}} c \text{ iff } h(s) \Rightarrow_{\hat{D}} c. & \\ s \xrightarrow{c|p}_{\hat{S}_e} s' & \\ \text{iff } (\exists a, b, a \wedge b = c \text{ and } s \Rightarrow_{\hat{S}_{e_1}} a & \\ \text{and } s_2^* \xrightarrow{b|p}_{\hat{S}_{e_2}} s') & \\ \text{or } (s \xrightarrow{c|p}_{\hat{S}_{e_1}} s') & \\ \text{iff } (\exists a, b, a \wedge b = c \text{ and } h_1(s) \Rightarrow_{\hat{D}} a & \\ \text{and } e_2 \xrightarrow{b|p}_{\hat{D}} h_2(s')) & \\ \text{or } (h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s')) & \\ \text{iff } h_1(s) \xrightarrow{c|p}_{\hat{D}} h(s') \text{ iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s'). & \end{aligned}$$

The case where  $s_2 \in \hat{S}_{e_2}$  is straightforward, as  $\hat{S}_e$  preserves the transitions of  $\hat{S}_{e_2}$ :

$$\begin{aligned} s \Rightarrow_{\hat{S}_e} c &\text{ iff } s \Rightarrow_{\hat{S}_{e_2}} c \\ \text{iff } h_2(s) \Rightarrow_{\hat{D}} c \text{ iff } h(s) \Rightarrow_{\hat{D}} c, & \end{aligned}$$

$$s \xrightarrow{c|p}_{\hat{S}_e} s' \text{ iff } s \xrightarrow{c|p}_{\hat{S}_{e_2}} s'$$

$$\text{iff } h_2(s) \xrightarrow{c|p}_{\hat{D}} h_2(s') \text{ iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s').$$

When  $e \triangleq e_1^{(b)}$ , by induction hypothesis, we have a homomorphism  $h_1 : \hat{S}_{e_1} \rightarrow \hat{D}$ ; the homomorphism  $h$  can be defined as follows:

$$h(s) \triangleq \begin{cases} e_1^{(b)} & s \triangleq s^* \\ h_1(s); e_1^{(b)} & s \in \hat{S}_{e_1} \end{cases}$$

We prove the homomorphism condition by case analysis on  $s$ . First case is that  $s = s^*$ , then:

$$(s^* \Rightarrow_{\hat{S}_e} c) \text{ iff } (s^* \Rightarrow_{\hat{S}_e} c \text{ and } c = \bar{b})$$

$$\text{iff } (e_1^{(b)} \Rightarrow_{\hat{D}} c \text{ and } c = \bar{b})$$

$$\text{iff } (h(s^*) \Rightarrow_{\hat{D}} c);$$

$$(s^* \xrightarrow{c|p}_{\hat{S}_e} s') \text{ iff } (\exists a, b \wedge a = c \text{ and } s_1^* \xrightarrow{a|p}_{\hat{S}_{e_1}} s')$$

$$\text{iff } (\exists a, b \wedge a = c \text{ and } e_1 \xrightarrow{a|p}_{\hat{D}} h_1(s'))$$

$$\text{iff } e_1^{(b)} \xrightarrow{a|p}_{\hat{D}} h_1(s') \text{ iff } h(s^*) \xrightarrow{a|p}_{\hat{D}} h(s')$$

The second case is when  $s \in \hat{S}_{e_1}$ , then:

$$s \Rightarrow_{\hat{S}_e} c$$

$$\text{iff } (\exists a, \bar{b} \wedge a = c \text{ and } s \Rightarrow_{\hat{S}_{e_1}} a)$$

$$\text{iff } (\exists a, \bar{b} \wedge a = c \text{ and } h_1(s) \Rightarrow_{\hat{D}} a)$$

$$\text{iff } h_1(s); e_1^{(b)} \Rightarrow_{\hat{D}} c \text{ iff } h(s) \Rightarrow_{\hat{D}} c$$

$$s \xrightarrow{c|p}_{\hat{S}_e} s'$$

$$\text{iff } (s \xrightarrow{c|p}_{\hat{S}_{e_1}} s'$$

$$\text{or } \exists a_1, a_2, b \wedge a_1 \wedge a_2 = c$$

$$\text{and } s \Rightarrow_{\hat{S}_{e_1}} a_1$$

$$\text{and } s_1^* \xrightarrow{a_2|p}_{\hat{S}_{e_1}} s')$$

$$\text{iff } (h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s')$$

$$\text{or } \exists a_1, a_2, b \wedge a_1 \wedge a_2 = c$$

$$\text{and } h_1(s) \Rightarrow_{\hat{D}} a_1$$

$$\text{and } e_1 \xrightarrow{a_2|p}_{\hat{D}} h_1(s'))$$

$$\text{iff } (h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s')$$

$$\text{or } \exists a_1, a_2, b \wedge a_1 \wedge a_2 = c$$

$$\text{and } h_1(s) \Rightarrow_{\hat{D}} a_1$$

$$\text{and } e_1^{(b)} \xrightarrow{b \wedge a_2|p}_{\hat{D}} h_1(s'); e_1^{(b)})$$

$$\text{iff } (h_1(s); e_1^{(b)} \xrightarrow{c|p}_{\hat{D}} h_1(s'); e_1^{(b)})$$

$$\text{iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s'). \quad \square$$

Theorem 20 have several consequences, one of the more obvious one is that we can use the functoriality of the lowering operation to show the semantic

equivalence of the start state in the thompson's construction and the expression in derivative.

**Corollary 21** (Correctness). *Given any expression  $e$  and its Thompson's coalgebra  $\hat{S}_e$  with the start state  $s^* \in \hat{S}_e$ , then the semantics of the start state is equivalent to the semantics of  $e$ :  $\llbracket s^* \rrbracket_{\hat{S}_e} = \llbracket e \rrbracket$ .*

*Proof.* By functoriality of lowering theorem 17, then  $h : S_e \rightarrow D$  is a homomorphism on their lowerings. By standard theorem of universal coalgebra, homomorphism preserves the result to the final coalgebra [6], in particular,  $\llbracket s^* \rrbracket_{\hat{S}_e}^\omega = \llbracket h(s^*) \rrbracket_D^\omega = \llbracket e \rrbracket_D^\omega$ , where  $\llbracket - \rrbracket^\omega$  is the infinite trace semantics i.e. the unique map into the final GKAT coalgebra.

Finally, because infinite trace equivalence implies finite trace equivalence (Corollary 12),  $\llbracket s^* \rrbracket_{S_e} = \llbracket e \rrbracket_D$ ; then the correctness of derivative (Theorem 19) concludes our proof:  $\llbracket s^* \rrbracket_{S_e} = \llbracket e \rrbracket_D = \llbracket e \rrbracket$ .  $\square$

A not so obvious consequence of the homomorphism in Theorem 20, is the complexity of the algorithm based on derivatives. Our bisimulation algorithm (Algorithm 2) only explores the principle sub-coalgebra of the start state, i.e.  $s^*$  in the Thompson's construction  $\hat{S}_e$  or  $e$  in the derivative  $\hat{D}$ ; thus, deducing an upper bound on the size of the principle sub-coalgebras  $\langle s^* \rangle_{\hat{S}_e}$  and  $\langle e \rangle_{\hat{D}}$  are crucial to our complexity analysis. An upper bound on  $\langle s^* \rangle_{\hat{S}_e}$  is easy to obtain, as the size of  $\hat{S}_e$ , which subsumes the states of  $\langle s^* \rangle_{\hat{S}_e}$ , is linear to the size of expression  $e$ ; therefore  $\langle s^* \rangle_{\hat{S}_e}$  is at most linear to the size of the expression  $e$ . On the other hand the size of  $\langle e \rangle_{\hat{D}}$  can, again, be derived from the homomorphism in theorem 20.

**Corollary 22.** *There exists a surjective homomorphism  $h' : \langle s^* \rangle_{\hat{S}_e} \rightarrow \langle e \rangle_{\hat{D}}$ . Because the size of  $\langle s^* \rangle_{\hat{S}_e}$  is linear to  $e$ , the size of  $\langle e \rangle_{\hat{D}}$  is at most linear to the size of expression  $e$ .*

*Proof.* We define  $h'$  to be point-wise equal to  $h$ , i.e.  $h'(s) \triangleq h(s)$ , i.e.  $h'$  is  $h$  restricted on the domain  $\langle s^* \rangle_{\hat{S}_e}$ . We need to show that  $h'$  is well-defined and surjective, which is a consequence of homomorphic image preserves principle sub-coalgebra (Theorem 2):  $h(\langle s^* \rangle_{\hat{S}_e}) = \langle h(s) \rangle_{\hat{D}} = \langle e \rangle_{\hat{D}}$ . In other words, the image of  $h$  on  $\langle s^* \rangle_{\hat{S}_e}$  is equal to  $\langle e \rangle_{\hat{D}}$ ; thus, because  $h'$  the restriction of  $h$  on  $\langle s^* \rangle_{\hat{S}_e}$ , the range of  $h'$  contains its codomain  $\langle e \rangle_{\hat{D}}$ , showing that  $h'$  is surjective.  $\square$

Therefore,  $\langle s^* \rangle_{\hat{S}_e}$  will have no less state than  $\langle e \rangle_{\hat{D}}$  as a consequence of the surjectivity of  $h'$  in corollary 22. Because  $\hat{S}_e$  have more states than  $\langle s^* \rangle_{\hat{S}_e}$ , which have more states than  $\langle e \rangle_{\hat{D}}$ , and notice that the size of  $\hat{S}_e$  is linear to the size of the input expression,

therefore  $\langle e \rangle_{\hat{D}}$  is at most linear to the size of the expression.

Although the size of the coalgebra generated by the derivative is smaller than Thompson’s construction, this does not imply that the decision procedure based on derivative is always more efficient than those based on Thompson’s construction. Crucially, the states in the derivative coalgebra  $\hat{D}$  are expressions, and computing the next transition of the coalgebra can also be more expensive. Thompson’s construction, on the other hand, only requires inductively going through the expression once to construct the entire coalgebra  $\hat{S}_e$ , and its states can be represented by more efficient constructs, like integers.

## VI. IMPLEMENTATION

### A. Optimization

### B. Performance

## VII. FUTURE WORK

Can weak symbolic coalgebra leads to a simpler completeness proof.

## REFERENCES

- [1] Ricardo Almeida, Sabine Broda, and Nelma Moreira. “Deciding KAT and Hoare Logic with Derivatives”. In: *Electronic Proceedings in Theoretical Computer Science* 96 (Oct. 2012), pp. 127–140. ISSN: 2075-2180. DOI: 10.4204/EPTCS.96.10. (Visited on 12/08/2023).
- [2] Bart Jacobs. “Introduction to Coalgebra: Towards Mathematics of States and Observation”. In: Cambridge University Press, Oct. 2016. ISBN: 978-1-107-17789-5 978-1-316-82318-7. DOI: 10.1017/CBO9781316823187. (Visited on 05/20/2024).
- [3] Dexter Kozen. “On the Coalgebraic Theory of Kleene Algebra with Tests”. In: *Rohit Parikh on Logic, Language and Society*. Ed. by Can Başkent, Lawrence S. Moss, and Ramswamy Ramanujam. Outstanding Contributions to Logic. Cham: Springer International Publishing, 2017, pp. 279–298. ISBN: 978-3-319-47843-2. DOI: 10.1007/978-3-319-47843-2\_15. (Visited on 01/17/2024).
- [4] Dexter Kozen and Frederick Smith. “Kleene Algebra with Tests: Completeness and Decidability”. In: *Computer Science Logic*. Ed. by Gerhard Goos et al. Vol. 1258. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 244–259. ISBN: 978-3-540-63172-9 978-3-540-69201-0. DOI: 10.1007/3-540-63172-0\_43. (Visited on 03/16/2021).
- [5] Damien Pous. “Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests”. In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’15. New York, NY, USA: Association for Computing Machinery, Jan. 2015, pp. 357–368. ISBN: 978-1-4503-3300-9. DOI: 10.1145/2676726.2677007. (Visited on 12/07/2023).
- [6] J. J. M. M. Rutten. “Universal Coalgebra: A Theory of Systems”. In: *Theoretical Computer Science*. Modern Algebra 249.1 (Oct. 2000), pp. 3–80. ISSN: 0304-3975. DOI: 10.1016/S0304-3975(00)00056-6.
- [7] Todd Schmid et al. *Guarded Kleene Algebra with Tests: Coequations, Coinduction, and Completeness*. May 2021. DOI: 10.4230/LIPIcs.ICALP.2021.142. arXiv: 2102.08286 [cs]. (Visited on 07/03/2023).
- [8] Steffen Smolka et al. “Guarded Kleene Algebra with Tests: Verification of Uninterpreted Programs in Nearly Linear Time”. In: *Proceedings of the ACM on Programming Languages* 4.POPL (Jan. 2020), pp. 1–28. ISSN: 2475-1421. DOI: 10.1145/3371129.

## APPENDIX