Symbolic On-the-fly Algorithms for GKAT Equivalences

Cheng Zhang[§]
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 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 $\delta : X \to Y \to Z$, we will write the function applications as follow $\delta(x) : Y \to Z$ and $\delta(x, y) : Z$. And when drawing commutative diagram, we will leave function restriction implicit. Specifically given $A' \subseteq A$, and a function $h : A \to B$, we will draw:

$$A' \xrightarrow{h} B$$

where the function h is implicitly restricted to A'. For bifunctors like $(-) \times (-)$, we will write function lifting by applying the bifunctors on these functions: for example, given $h_1 : A_1 \to B_1$ and $h_2 : A_2 \to B_2$, we will use

$$h_1 \times h_2 : A_1 \times A_2 \to B_1 \times B_2$$

to denote the bifunctorial lift of h_1 and h_2 via product $(-) \times (-)$.

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 $\sigma_S : S \to F(S)$. 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 $h : S \to U$ that preserves the function σ; diagrammatically, the following diagram commutes:

$$\begin{tikzcd} S \arrow[r, "h"] \arrow[d, "\sigma_S"'] & U \arrow[d, "\sigma_U"] \\ F(S) \arrow[r, "F(h)"] & F(U) \end{tikzcd}$$

When we can restrict the homomorphism map into a inclusion map $i : S' \to S$ 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 $S' \sqsubseteq S$:

$$\begin{tikzcd} S' \arrow[r, hook, "i"] \arrow[d, "\sigma_{S'}"'] & S \arrow[d, "\sigma_S"] \\ F(S') \arrow[r, hook, "F(i)"] & F(S) \end{tikzcd}$$

In fact, the function $\sigma_{S'}$ is uniquely determined by the states S' [7, Proposition 6.1].

The sub-coalgebras are preserved under homomorphic images and pre-images:

Lemma 1 (Theorem 6.3 [7])**.** *given a homomorphism $h : S \to U$, and sub-coalgebras $S' \sqsubseteq S$ and $U' \sqsubseteq U$, then*

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

Identify applicable funding agency here. If none, delete this.
[§]Work largely performed at Boston University

One particularly important sub-coalgebra of and coalgebra S is the least coalgebra generated by a single element 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" form the state s. This coalgebraic characterization of reachable state can allow us to avoid induction on the length of path from s to reach another state.

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 [7, 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 $h : S \to U$:*

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

Proof. We will need to show that $h(\langle s \rangle_S)$ is the smallest sub-GKAT coalgebra of S that contain $h(s)$. First by definition of image, $h(s) \in h(\langle s \rangle_S)$; second by lemma 1, $h(\langle S \rangle_S) \sqsubseteq U$.

Finally, take any $U' \sqsubseteq U$ and $h(s) \in U'$, recall that by lemma 1, $h^{-1}(U') \sqsubseteq S$. We can then derive that $h(\langle s \rangle_S) \sqsubseteq U'$:

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

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

A *final coalgebra* $\mathcal{F}$ over a signature F, sometimes called the *behavior* of coalgebras over F, is a F-coalgebra s.t. for all F-coalgebra S, there exists a unique homomorphism $[\![-]\!]_S : S \to \mathcal{F}$.

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:

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

And its projections $\pi_1 : S \times U \to S$ and $\pi_2 : S \times U \to U$ are both homomorphisms:

$$\begin{tikzcd} S \arrow[d, "\sigma_S"'] & \sim \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\sigma_\sim"] & U \arrow[d, "\sigma_T"] \\ F(S) & F(\sim) \arrow[l, "F(\pi_1)"] \arrow[r, "F(\pi_2)"'] & F(T) \end{tikzcd}$$

In a special case, when there exists a homomorphism $h : S \to U$, 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 $\sigma_\sim \triangleq \sigma_S \times \sigma_U$.

Corollary 3. *Given two F-coalgebra S, U and a homomorphism $h : S \to U$, then all for all s, $[\![s]\!]_S = [\![h(s)]\!]_U$.*

Finally, we are interested in bisimulation equivalence. Given a coalgebra S, a bisimulation $\simeq \subseteq S \times S$ is a bisimulation equivalence when $\simeq$ is a bisimulation and also an equivalence relation. As it turns out, searching for bisimulation is equivalent to bisimulation equivalence. This result is very important, as it allows us to leverage more efficient data structure, like union-find, to search for equivalence relation instead of relations in general.

Theorem 4. *For any coalgebra S, there exists a bisimulation $\sim \subseteq S \times S$ s.t. $s_1 \sim s_2$ if and only if there exists a bisimulation equivalence $\simeq \subseteq S \times S$ s.t. $s_1 \simeq s_2$*

Proof. First show the $\implies$ direction, we consider the maximal bisimulation between S and itself $\equiv \subseteq S \times S$, which is known to be a bisimulation equivalence [7, Corollary 5.6]. Because $\equiv$ is maximal, therefore $\sim \subseteq \equiv$ and $s_1 \equiv s_2$.

Then the $\impliedby$ direction can be proven by setting $\sim \triangleq \simeq$. ☐

B. GKAT and Its Coalgebra

Guarded Kleene Algebra with Tests, or GKAT [9], 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 Bool and expressions Exp:

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

where $e +_b f$ is the if statement with condition b; $e; f$ is the sequencing of expression e and f; finally $e^{(b)}$ is the while loop with body e and condition b. A GKAT expression can be unfolded into a KAT expression in the usual manner [5]:

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

Then the semantics of each expression $[\![e]\!]$ can be computed by the semantics of Kleene Algebra with tests [5]. An important construct in the semantics is *atoms*, which are a string of all the primitive tests either in its positive or negative form: for $B \triangleq \{b_1, b_2, ..., b_n\}$

$$\mathbf{At}_B \triangleq \{b'_1 \cdot b'_2 \cdot \dots \cdot b'_n \mid b'_i \in \{b_i, \overline{b_i}\}\}.$$

Follow the conventional notation, we denote an atom using α, β; and we sometimes omit the subscript B when no confusing can arise. Intuitively, atoms are truth assignments to each primitive tests, indicating which primitive tests is satisfied in the current program states.

For the sake of brevity, we omit the complete definition of GKAT and KAT semantics, we refer the reader to previous works [9, 8, 5], which explains these semantics in detail. Our work avoids directly interact with the semantics by leveraging 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:

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

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 α, denoted as $\delta(s, \alpha) = \mathrm{rej}$; accept α, denoted $\delta(s, \alpha) = \mathrm{acc}$; or transition to a state $s' \in S$ and execute action $p \in K$, denoted as $\delta(s, \alpha) = (s', p)$.

This deterministic behavior contrast that of Kleene coalgebra with tests [4], where for each atom, the state can accept or reject the atom (but not both), yet the state can also non-deterministically transition to multiple different state 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 [6], but also present challenges. Specifically, GKAT coalgebra requires normalization to compute finite trace equivalences [9], where we will remove all the state that cannot lead to acceptance, which are called "dead states".

In previous works, these dead states are detected after all the necessary state and transition is computed. This approach requires storing all the information of the coalgebra in memory, which do not allow on-the-fly computation, which can not only terminate whenever a counter-example is found, but can also erase past states from memory.

However, to truly understand the on-the-fly algorithm, 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 [9]. However, it is straightforward to show that principle sub-coalgebra $\langle s \rangle_S$ exactly correspondents to reachable states of s in coalgebra S. Thus, the classical definition is equivalent to the following:

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

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

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

Proof. $\Longleftarrow$ direction is true, because $s \in \langle s \rangle$: if all $\langle s \rangle$ is dead, then s is dead. $\Longrightarrow$ direction can be proven as follows. Take $s' \in \langle s \rangle$, then $\langle s' \rangle \sqsubseteq \langle s \rangle$ by definition. Since there is no accepting state in $\langle s \rangle$, thus there cannot be any accepting state in $\langle s' \rangle$, hence $\langle s' \rangle$ is also dead. ☐

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

$$s \textit{ is live} \Longleftrightarrow h(s) \textit{ is live}$$

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 $s' \in S$:

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

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 $h(\langle s \rangle_S) = \langle h(s) \rangle_U$ contains an accepting state. Therefore, s is live in S if and only if $h(s)$ is live in U. ☐

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

Corollary 7 (sub-coalgebra perserves 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 $i : S' \to S$. ☐

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 $\pi_1 : \sim \to S$ and $\pi_2 : \sim \to U$ are homomorphisms. Therefore,

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

D. Normalization And Semantics

(Possibly infinite) trace model $\mathcal{G}_\omega$ is the final coalgebra of GKAT coalgebras [8]. The finality of the model implies that every state in any GKAT coalgebra S can be assigned a semantics under the unique homomorphism $[\![-]\!]^\omega_S : S \to \mathcal{G}_\omega$; and such semantic equivalences can indeed be identified by bisimulation [8]: $[\![s]\!]^\omega_S = [\![t]\!]^\omega_T$ 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 [4, 1, 6]. However, the *finite* trace model $\mathcal{G}$ is the final coalgebra of GKAT coalgebras without dead states, which we call *normal GKAT coalgebra* [9]. Fortunately every GKAT coalgebra can be normalized by rerouting all the transition from dead states to rejection. Concretely, given a GKAT coalgebra $S \triangleq (S, \delta_S)$, we let $\delta_{\mathrm{norm}(S)} : S \to G(S)$ is defined as $\delta_{\mathrm{norm}(S)}(s,\alpha) \triangleq \mathrm{rej}$ when $\delta_S(s,\alpha) = (s',p)$ and s' is dead in S; and $\delta_{\mathrm{norm}(S)}(s,\alpha) \triangleq \delta_S(s,\alpha)$ otherwise; then $(S, \delta_{\mathrm{norm}(S)})$ is the *normalized* coalgebra of $S \triangleq (S, \delta_S)$ denoted as $\mathrm{norm}(S)$.

The finality of $\mathcal{G}$ means that the finite trace semantics $[\![-]\!]$ is the unique coalgebra homomorphism $\mathrm{norm}(S) \to \mathcal{G}$. Furthermore, the finite trace equivalence between $s \in S$ and $u \in U$ can be computed by first normalizing S and U, then decide whether there is a bisimulation on $\mathrm{norm}(S)$ and $\mathrm{norm}(U)$ that includes (s,t). For a more detailed explanation on the finite trace semantics, we refer the reader to the work of Smolka et al. [9].

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 9. norm *is an endofunctor in the category GKAT coalgebra. More specifically, if $h : S \to U$ is a GKAT homomorphism, then $h : \mathrm{norm}(S) \to \mathrm{norm}(U)$ 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 $r \in \{\mathrm{rej}, \mathrm{acc}\}$,
$$\delta_S(s,\alpha) = r \iff \delta_U(h(s),\alpha) = r;$$
- 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 $h : \mathrm{norm}(S) \to \mathrm{norm}(U)$ is a homomorphism, this is a consequence of homomorphism preserves liveness (Theorem 6): for all $s \in \mathrm{norm}(S)$ and $\alpha \in \mathbf{At}$:

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

Corollary 10. *Normalization preserves sub-coalgebra, i.e. if $S' \sqsubseteq S$ then $\mathrm{norm}(S') \sqsubseteq \mathrm{norm}(S)$.*

Proof. By letting the homomorphism in Theorem 9 to be the inclusion homomorphism $i : S' \to S$ □

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

Corollary 11. *Given two states in two GKAT coalgebra $s \in S, u \in U$, $[\![s]\!]^\omega_S = [\![u]\!]^\omega_U \implies [\![s]\!]_S = [\![u]\!]_U$.*

Proof. Because $[\![s]\!]^\omega_S = [\![u]\!]^\omega_U$, there exists a bisimulation $\sim \subseteq S \times U$ s.t. $s \sim u$ [8]. Therefore, we have the following span in the category of GKAT coalgebra:

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

Then by Theorem 9, $\mathrm{norm}(\sim)$ is a bisimulation between $\mathrm{norm}(S)$ and $\mathrm{norm}(U)$:

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

And because normalization operation preserves states in $\sim$, therefore $(s,u) \in \mathrm{norm}(\sim)$, and because $\mathrm{norm}(\sim)$ is a bisimulation between the normalization of S and U, therefore $[\![s]\!]_S = [\![u]\!]_U$ □

III. On-The-Fly Bisimulation

The original algorithm for deciding GKAT equivalences [9] requires the entire automaton 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 normalized the automaton only when we need to.

In this section, we introduce an algorithm that merges bisimulation and normalization where we only need to test the liveness of the state when a disparity in the bisimulation has been found. For example, when one automaton leads to reject where the other transition to a state, then we will need to verify whether that state is dead or not.

This on-the-fly algorithm inherits the efficiency of the original algorithm [9], where the worst case will require two passes of the automaton, where one pass will try to establish a bisimulation, when failed the other pass will kick in and compute whether the failed states are dead. In some special case, the on-the-fly algorithm can even out perform the original algorithm; for example, when the two input automata are bisimular (even when they are not normal), the on-the-fly algorithm can skip the liveness checking, only performing the bisimulation.

TODO: I think we should move the next couple theorem to the background.

Theorem 12 (sub-coalgebras perserve and reflect bisimulation)**.** *Given any sub-coalgebra $S' \sqsubseteq S$ and $T' \sqsubseteq T$,*

- *Given a bisimulation $\sim$ between S' and T', then $\sim$ is also a bisimulation between S and T;*
- *if there exists a bisimulation $\sim$ between S and T, then the restriction*

$$\sim_{S',T'} \triangleq \{(s,t) \mid s \in S', t \in T', s \sim t\}$$

forms a bisimulation between S' and T'.

Proof. To prove that bisimulation $\sim$ between S' and T' is also a bisimulation of S and T, we can simply enlarge the diagram by the inclusion homomorphism

$$\begin{tikzcd}
S \arrow[d, "\delta_S"] & S' \arrow[l, hook', "i"'] \arrow[d, "\delta_S"] & \sim \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\delta_\sim"] & T' \arrow[r, hook, "i"] \arrow[d, "\delta_S"] & T \arrow[d, "\delta_T"] \\
G(S) & G(S') \arrow[l, hook', "G(i)"'] & G(\sim) \arrow[l, "G(\pi_1)"'] \arrow[r, "G(\pi_2)"] & T' \arrow[r, hook, "G(i)"] & T
\end{tikzcd}$$

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

$$\sim \xrightarrow{\pi_1} S' \xrightarrow{i} S = \sim \xrightarrow{\pi_1} S \qquad \sim \xrightarrow{\pi_2} T' \xrightarrow{i} T = \sim \xrightarrow{\pi_2} T$$

To prove that the bisimulation can be restricted, we first realize that $\sim_{S',T'}$ is a pre-image of the maximal bisimulation $\equiv_{S',T'}$ along the inclusion homomorphism $i : \sim \to \equiv_{S,T}$. This means that $\sim_{S',T'}$ can be formed by a pullback square:

$$\begin{tikzcd}
\sim_{S',T'} \arrow[r, "i"] \arrow[d, "i"'] \arrow[dr, phantom, "\lrcorner", very near start] & \equiv_{S',T'} \arrow[d, "i"] \\
\sim \arrow[r, "i"'] & \equiv_{S,T}
\end{tikzcd}$$

Recall that elementary polynomial functor [3] like G preserves pullback, hence the pullback also uniquely generates a GKAT coalgebra [7] $\square$

Theorem 12 allows us to only search for bisimulation on a sub-coalgebra, speeding up our search algorithm. Another way to speed up the algorithm is to use efficient data structures to find a bisimulation equivalence instead of a bisimulation relation. This optimization is a special case of the *up-to technique* [2]. Specifically, we will extend Theorem 4 to a setting where the bisimulation is no longer over the same coalgebra.

Theorem 13. *Given two states in two coalgebras $s \in S, u \in U$, let $S+U$ be the coproduct coalgebra of S and U [7]. There exists a bisimulation $\sim \subseteq S \times U$ s.t. $s \sim u$ if and only if there exists a bisimulation equivalence on $\simeq \subseteq (S+U) \times (S+U)$, s.t. $s \simeq u$.*

Proof. Notice that both S and U are sub-coalgebra of $S+U$ witnessed by the canonical injection $\mathrm{inj}_l : S \to S+U$ and $\mathrm{inj}_r : U \to S+U$.

Let $\sim$ be a bisimulation and $\simeq$ be a bisimulation equivalence:

$$\begin{aligned}
&\exists \sim \subseteq S \times U, s \sim u \\
\iff &\exists \sim \subseteq (S+U) \times (S+U), s \sim u \\
\iff &\exists \simeq \subseteq (S+U) \times (S+U), s \simeq u.
\end{aligned}$$ $\square$

After we justify the above optimizations of the algorithm, we will show the core theorem that establishes the correctness of our algorithm.

Lemma 14 (bisimulation between dead states)**.** *Given two dead states $s \in S$ and $u \in U$, then the singleton bisimulation $\sim \subseteq \mathrm{norm}(S) \times \mathrm{norm}(U)$:*

$$\sim \triangleq \{(s,u)\} \qquad \delta_\sim((s,u),\alpha) \triangleq \mathrm{rej}$$

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

Proof. By computation $\square$

Theorem 15 (Recursive Construction). *Given two GKAT coalgebra S and U, and two of their elements $s \in S$ and $u \in U$, there exists a bisimulation $\sim \subseteq \mathrm{norm}(\langle s\rangle) \times \mathrm{norm}(\langle u\rangle)$ s.t. $s \sim u$, if and only if all the following holds:*

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

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

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

The condition 1 holds:

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

The condition 2 holds:

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

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 $\mathrm{norm}(\delta_S)(s', \alpha)$ can either be rejection or (s', p), and so is $\mathrm{norm}(\delta_U)(u', \alpha)$:

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

- If both s' and u' are live, then $s' \sim u'$. By theorem 12, 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 14.

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 π_1 but projects to (t', q) under π_2. Thus both s' and t' has to be dead.

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

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

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 10, $\mathrm{norm}(\langle s'\rangle) \sqsubseteq \mathrm{norm}(\langle s\rangle)$ and $\mathrm{norm}(\langle u'\rangle) \sqsubseteq \mathrm{norm}(\langle u\rangle)$. By Theorem 12, every $\equiv_{s',u'}$ is a bisimulation between $\mathrm{norm}(\langle s\rangle)$ and $\mathrm{norm}(\langle t\rangle)$, and because bisimulation is closed under arbitrary union [7], $\sim'$ is a bisimulation between $\mathrm{norm}(\langle s\rangle)$ and $\mathrm{norm}(\langle t\rangle)$.

To obtain the desired bisimulation $\sim$ between $\mathrm{norm}(\langle s\rangle)$ and $\mathrm{norm}(\langle u\rangle)$, 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 $\delta_{\mathrm{norm}(S)}(s, \alpha) = \delta_{\mathrm{norm}(U)}(u, \alpha) = \mathrm{acc}$, then $\delta_{\sim}((s, u), \alpha) \triangleq \mathrm{acc}$;
- if $\delta_{\mathrm{norm}(S)}(s, \alpha) = \delta_{\mathrm{norm}(U)}(u, \alpha) = \mathrm{rej}$, then $\delta_{\sim}((s, u), \alpha) \triangleq \mathrm{rej}$;
- if $\delta_{\mathrm{norm}(S)}(s, \alpha) = (s', p)$ and $\delta_{\mathrm{norm}(U)}(u, \alpha) = (u', p)$, 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 $\mathrm{norm}(\langle s\rangle)$ and $\mathrm{norm}(\langle u\rangle)$ where $s \sim u$. We show the slightly more complicated case: $\delta_{\mathrm{norm}(S)}(s, \alpha) = (s', p)$ and $\delta_{\mathrm{norm}(U)}(u, \alpha) = (u', p)$ implies $s' \sim u'$ as an example. By condition 3, there exists a bisimulation $\sim_{s',u'} \subseteq \mathrm{norm}(\langle s'\rangle) \times \mathrm{norm}(\langle u'\rangle)$, and because $\equiv_{s',u'}$ is the maximal bisimulation between $\mathrm{norm}(\langle s'\rangle)$ and $\mathrm{norm}(\langle u'\rangle)$,

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

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 δ_S: for all $\alpha \in \mathbf{At}$,

- If $\delta_S(s, \alpha) = \mathrm{acc}$, then by condition 1, $\delta_U(u, \alpha) = \mathrm{acc}$; therefore
$$(\delta_{\mathrm{norm}(S)})(s, \alpha) = \mathrm{norm}(\delta_{\mathrm{norm}(U)})(u, \alpha) = \mathrm{acc}\,.$$
- 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
$$\delta_{\mathrm{norm}(S)}(s, \alpha) = \delta_{\mathrm{norm}(U)}(u, \alpha) = \mathrm{rej}\,.$$

- 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 $\delta_{\text{norm}(U)}(u, \alpha) = (u', p)$. ☐

Corollary 16 (Correctness)**.** *For any two states in two GKAT coalgebra $s \in S, u \in U$,the four condition in Theorem 15 is satisfied if and only if s and u are finite-trace equivalent*

Proof. By the standard argument with normalization preserves sub-coalgebra (Corollary 10), we can obtain $\text{norm}(\langle s \rangle) \sqsubseteq \text{norm}(S)$ and $\text{norm}(\langle u \rangle) \sqsubseteq \text{norm}(U)$. Because sub-coalgebra reflects and preserves bisimulation (Theorem 12) with the correctness of bisimulation between normalized coalgebras: let $\sim$ and $\sim'$ be bisimulations,

$$\begin{aligned} &\exists \sim \subseteq \text{norm}(\langle s \rangle) \times \text{norm}(\langle u \rangle), s \sim u \\ \iff &\exists \sim' \subseteq \text{norm}(S) \times \text{norm}(U), s \sim u \\ \iff &[\![s]\!]_S = [\![u]\!]_U. \end{aligned}$$

☐

The above theorem already gives us a way to recursively construct an algorithm that include $s \sim u$, this consequently will let us decide the trace equivalence of s and u: $[\![s]\!] = [\![u]\!]$. However, this algorithm can be further optimized, we will then derive that a dead state can never relate to live states. This means that when checking the bisimulation of states s and t, if we already know one of them is dead, we only need to check whether the other is dead, instead of going through the convoluted process mentioned in theorem 15.

However because homomorphism preserves liveness, if we already know one of the s and t is dead, the other has to be dead.

Theorem 17. *Given two states $s \in S$ and $t \in T$, if s is a dead state in S, then there exists a bisimulation $\sim$ between S and T where $s \sim t$ if and only if t is dead. Similarly for $t \in T$.*

Proof. if there exists a bisimulation $\sim$, s.t. $s \sim t$, because s is dead and bisimulation preserves liveness corollary 8, then t is dead.

And if both t and s is dead, then a bisimulation can by constructed by lemma 14. ☐

IV. The Algorithm

In this section we will present the pseudo-code for our on-the-fly algorithm. In order to implement the the inductive construction theorem (theorem 15), we will need to determine the liveness of the state. This can be simply computed via a DFS from the state being checked.

TODO: we should merge the two so that it is easier to

Algorithm 1 Check whether a state s is dead

---

function IsDeadLoop($s \in S$, explored)
  if $s \in$ explored **then return** explored
  else
    for $\alpha \in$ **At do**
      match $\delta_S(s, \alpha)$ **with**
        case acc **then return** none ▷ s transition to accept
        case rej **then** `continue` ▷ skip if s transition to reject
        case (s', p) **then**
          if IsDeadLoop(s') = none **then return** none ▷ s transitions to a live state s'
          else explored ← (explored ∪ IsDeadLoop(s', explored))
  return explored

By lemma 5, if s is dead then all the reachable states of s (denoted by $\langle s \rangle$). Then by returning all the reachable states of s, we can cache these states to avoid checking them again. To encapsulate the caching, we have the following function, which we will actually use in our bisimulation algorithm.

Algorithm 2 A cached algorithm to check whether a state is dead

---

deadStates ← ∅
function IsDead($s \in S$)
  if $s \in$ deadStates **then return** true
  else if IsDeadLoop($s, \emptyset$) = none **then return** false
  else
    deadStates ← (deadStates ∪ IsDeadLoop($s, \emptyset$))
    return true

Given the direct correspondence between bisimulation and bisimulation equivalence and bisimulation in sub-algebra:

$$\begin{aligned} &\exists \text{ bisimulation } \sim \subseteq \langle s \rangle \times \langle t \rangle \text{ s.t. } s \sim t \\ \iff &\exists \text{ bisimulation } \sim \subseteq (\langle s \rangle \cup \langle t \rangle) \times (\langle s \rangle \cup \langle t \rangle) \text{ s.t. } s \sim t \\ \iff &\exists \text{ bisimulation equivalence } \simeq \subseteq (\langle s \rangle \cup \langle t \rangle) \times (\langle s \rangle \cup \langle t \rangle) \text{ s.t. } s \simeq t \end{aligned}$$

we can safely replace the bisimulation in inductive construction (theorem 15) with bisimulation equivalence. Dealing with equivalence relations allows us to leverage efficient data structures like union find in our bisimulation algorithm.

We will use UNION(s,t) to denote the operation to equate s and t in a union-find, and use EQ(s,t) to check if s and t belongs to the same equivalence class, i.e. share the same representative. Specifically, we will use the union-find structures to keep track of the equivalence classes that we are in the process of checking, hence avoiding repeatedly checking the same pair of states to remove infinite loops.

Our on-the-fly bisimulation algorithm will decide whether there exists a bisimulation relation in $\langle s\rangle \cup \langle t\rangle$ s.t. $s \sim t$. This algorithm generally reproduce the setting of inductive construction theorem theorem 15; except by theorem 17, in the special case where s or t is dead, then we will only need to check whether the other is dead.

Because the dead state detection algorithm is coalgebra-specific, we use a subscript on "deadStates" and "IsDead" to indicate the coalgebra. The soundness and completeness of algorithm 3 can be observed by the fact that *when the algorithm terminate*, the algorithm returns true if and only if there exists a bisimulation between $\langle s\rangle$ and $\langle t\rangle$ s.t. $s \sim t$, which is then logically equivalent to trace equivalence. Such equivalence is a direct consequence of theorems 15 and 17.

Remark 2. *The caching of dead state and the short-cut to check whether s is dead when t is dead and vise versa, is not essential to the soundness and completeness of algorithm, they are here to trade speed with memory. In a memory-constraint situation, the "*deadStates*" variable can be cleared periodically to save memory.*

V. Symbolic Algorithm

Given the alphabet K, B, a *symbolic GKAT coalgebra* $\hat{S} \triangleq \langle S, \hat{\epsilon}, \hat{\delta}\rangle$ consists of a state set S and a accepting function $\hat{\epsilon}$ and a transition function $\hat{\delta}$:

$$\hat{\epsilon} : S \to \mathcal{P}(\mathsf{Bool}_B), \qquad \hat{\delta} : S \to \mathcal{P}(\mathsf{Bool}_B \times S \times K),$$

where Bool_B is the free boolean algebra over B (boolean expressions modulo boolean algebra axioms); for all states $s \in S$, all the booleans are "disjoint"; namely the conjunction of any two expression from the set $\{\hat{\epsilon}(s)\} \cup \{b \mid \exists (b, s', p) \in \delta(s)\}$ are false. We will then use $\hat{\rho}(s) : \mathsf{Bool}_B$ to denote the boolean expressions that contain all the atoms that the state s rejects, and $\hat{\rho}(s)$ can be computed as follows:

$$\hat{\rho}(s) \triangleq \neg\hat{\epsilon}(s) \vee \neg \left(\bigvee_{(b,s',p)\in\delta(s)} b \right)$$

Instead of modeling each atom individually in the automata, we group them into boolean expressions, this leads to a much more space efficient automata, and enables efficient bisimulation algorithms using off-the-shelf SAT solvers.

With the above intuition in mind, a symbolic GKAT coalgebra $\hat{S} \triangleq \langle S, \hat{\epsilon}, \hat{\delta}\rangle$ can be lowered into a GKAT coalgebra $\langle S, \delta\rangle$ in the following manner:

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

This is well-defined, i.e. no more than one clause can be satisfied precisely because the boolean expressions appear in $\hat{\epsilon}$ and $\hat{\delta}$ are disjoint. The trace semantics of a GKAT coalgebra $\langle S, \hat{\epsilon}, \hat{\delta}\rangle$ is then defined as the trace semantics of its lowering $\langle S, \delta\rangle$.

Remark 3 (Canonicity)**.** *Notice that symbolic GKAT coalgebra is not canonical, i.e. there exists two different symbolic GKAT colagebra with the same lowering, consider the state set $S \triangleq \{*\}$:*

$$\hat{\delta}_1(*) \triangleq \{b \mapsto (*, p), \neg b \mapsto (*, p)\} \qquad \hat{\delta}_2(*) \triangleq \{\top \mapsto (*, p)\},$$

and both $\hat{\epsilon}$ will return constant $\perp$. These two symbolic GKAT coalgebra obviously have the same lowering hence behavior, yet, they are different. There are other symbolic representation that will satisfy canonicity, yet we opt to use our current representation for ease of construction and computational efficiency.

Theorem 18 (Functoriality)**.** *The lowering operation is a functor, given a symbolic GKAT coalgebra homomorphism $h : \hat{S} \to \hat{U}$, then h is also a homomorphism $h : S \to U$.*

Proof. □

We can then migrate the normalized bisimulation algorithm to the symbolic setting, we will first prove an inductive construction theorem like theorem 15.

Theorem 19 (Symbolic Inductive Construction)**.** *Given two symbolic GKAT coalgebra $\hat{S} = \langle S, \hat{\epsilon}_S, \hat{\delta}_S\rangle$ and $\hat{T} = \langle T, \hat{\epsilon}_T, \hat{\delta}_T\rangle$ and two states $s \in S$ and $t \in T$, there exists a normalized bisimulation on the lowered coalgebra $\sim \subseteq S \times T$ s.t. $s \sim t$ if and only if all the following holds:*

- $\bigvee \hat{\epsilon}_S(s) \equiv \bigvee \hat{\epsilon}_T(t)$*;*
- *for all $(b, s', p) \in \hat{\delta}_S(s)$ and $(c, t', q) \in \hat{\delta}_T(t)$, if $b \wedge c \not\equiv 0$ and $p = q$ then here exists a normalized bisimulation $\sim_{s',t'} \subseteq S \times T$ s.t. $s' \sim_{s',t'} t'$;*
- *for all $(b, s', p) \in \hat{\delta}_S(s)$ and $(c, t', q) \in \hat{\delta}_T(t)$, if $b \wedge c \not\equiv 0$ and $p \neq q$ then both s' and t' is dead;*

Algorithm 3 On-the-fly bisimulation algorithm

```
function EQUIV(s ∈ S, t ∈ T)
    if EQ(s, t) then return true
    else if s ∈ deadStates_S then return ISDEAD_T(t)
    else if t ∈ deadStates_T then return ISDEAD_S(s)
    else
        for α ∈ At do                                    ▷ Inductive construction, theorem 15
            match δ_S(s, α), δ_T(t, α) with
                case acc , acc then skip
                case rej , rej then skip
                case rej , (t′, q) then ISDEAD(t′)
                case (s′, p), rej then ISDEAD(s′)
                case (s′, p), (t′, q) then
                    if p = q then UNION(s, t); EQUIV(s, t)
                    else if ISDEAD(s) and ISDEAD(s) then skip
                    else return false
                default return false                     ▷ the results format does not match
    return true                                          ▷ no mismatch found
```

- *for all* $(b, s', p) \in \hat{\delta}_S(s)$ *and* $c \in \hat{\rho}_T(t)$*, if* $b \wedge c \not\equiv 0$*, then* s' *is dead;*
- *for all* $b \in \hat{\rho}_S(s)$ *and* $(c, t', q) \in \hat{\delta}_T(t)$*, if* $b \wedge c \not\equiv 0$*, then* t' *is dead;*

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 lowered coalgebra. □

Then for the algorithm, we can just recursively check all the conditions in theorem 19.

Inspired by the syntax of Ocaml, the && is the logical-and operator on the language level, specifying that all four conditions in the return statements must be satisfied to return true. Notice just like the non-symbolic case, this algorithm can be modified to perform symbolic bisimulation of (non-normalized) GKAT automaton, which coincides with infinite trace equivalence [8], by letting IsDead always return false and keep deadStates empty.

VI. Construction of Symbolic GKAT Automata

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 [8, 9]. 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. [8]. 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 20 (Correctness)**.** *The lowering of* $\hat{D}$ *is exactly the derivative defined by Schmid et. al. [8]. TODO: unfold the statement.*

Another way to construct an automaton 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, like in if statements, the guard expression can be used to simulate uniform continuation. Specifically, we can use $\langle\hat{\epsilon}(s)|\ \delta(s)$ 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 $(S_1, \hat{\epsilon}_1, \hat{\delta}_1)$ and $(S_2, \hat{\epsilon}_2, \hat{\delta}_2)$ 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

Algorithm 4 Symbolic On-the-fly bisimulation algorithm

function $\textsc{equiv}(s \in S, t \in T)$
  if $\textsc{eq}(s,t)$ **then return** true
  else if $s \in \text{deadStates}_S$ **then return** $\textsc{isDead}_T(t)$
  else if $t \in \text{deadStates}_T$ **then return** $\textsc{isDead}_S(s)$
  else return ▷ conditions of theorem 19

$$\bigvee \hat{\epsilon}_S(s) \equiv \bigvee \hat{\epsilon}_T(t) \ \&\&$$

$$\forall (b, s', p) \in \hat{\delta}_S(s), (c, t', q) \in \hat{\delta}_T(t), (b \wedge c) \not\equiv \bot \implies \begin{cases} \textsc{IsDead}_S(s) \wedge \textsc{IsDead}_T(t) & \text{if } p \neq q \\ \textsc{Union}(s,\ t); \textsc{Equiv}(s', t') & \text{if } p = q \end{cases} \ \&\&$$

$$\forall (b, s', p) \in \hat{\delta}_S(s), c \in \hat{\rho}_T(t), (b \wedge c) \not\equiv \bot \implies \textsc{IsDead}_S(s') \ \&\&$$

$$\forall b \in \hat{\rho}_S(s), (c, t', q) \in \hat{\delta}_T(t), (b \wedge c) \not\equiv \bot \implies \textsc{IsDead}_T(t')$$

$$\frac{}{p \xrightarrow{1|p}_{\hat{D}} 1} \qquad \frac{}{b \Rightarrow_{\hat{D}} b} \qquad \frac{e \xrightarrow{c|p}_{\hat{D}} e'}{e +_b f \xrightarrow{b \wedge c|p}_{\hat{D}} e'} \qquad \frac{e \Rightarrow_{\hat{D}} c}{e +_b f \Rightarrow_{\hat{D}} b \wedge c} \qquad \frac{f \xrightarrow{c|p}_{\hat{D}} f'}{e +_b f \xrightarrow{\overline{b} \wedge c|p}_{\hat{D}} f'} \qquad \frac{f \Rightarrow_{\hat{D}} c}{e +_b f \Rightarrow_{\hat{D}} \overline{b} \wedge c}$$

$$\frac{e \Rightarrow_{\hat{D}} b \qquad f \Rightarrow_{\hat{D}} c}{e; f \Rightarrow_{\hat{D}} b \wedge c} \qquad \frac{e \xrightarrow{b|p}_{\hat{D}} e'}{e; f \xrightarrow{b|p}_{\hat{D}} e'; f} \qquad \frac{e \Rightarrow_{\hat{D}} b \qquad f \xrightarrow{c|p}_{\hat{D}} f'}{e; f \xrightarrow{b \wedge c|p}_{\hat{D}} f'} \qquad \frac{}{e^{(b)} \Rightarrow_{\hat{D}} \overline{b}} \qquad \frac{e \xrightarrow{c|p} e'}{e^{(b)} \xrightarrow{b \wedge c|p}_{\hat{D}} e'; e^{(b)}}$$

Fig. 1: Symbolic Derivative of GKAT Automata.

Exp	S	s^*	$\hat{\epsilon}(s)$	$\hat{\delta}(s)$
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^*	$\begin{cases} \langle\{b\} \vert \hat{\epsilon}_1(s_1^*) \cup \langle\{b\} \vert \hat{\epsilon}_2(s_2^*) & s = s^* \\ \hat{\epsilon}_1(s) & s \in S_1 \\ \hat{\epsilon}_2(s) & s \in S_2 \end{cases}$	$\begin{cases} \langle\{b\} \vert \hat{\delta}_1(s_1^*) + \langle\{b\} \vert \hat{\delta}_2(s_2^*) & s = s^* \\ \hat{\delta}_1(s) & s \in S_1 \\ \hat{\delta}_2(s) & s \in S_2 \end{cases}$
$e_1; e_2$	$S_1 + S_2$	s_1^*	$\begin{cases} \langle\hat{\epsilon}_1(s) \vert \hat{\epsilon}_2(s_2^*) & s \in S_1 \\ \hat{\epsilon}_2(s) & s \in S_2 \end{cases}$	$\begin{cases} \hat{\delta}_1(s) + \langle\hat{\epsilon}(s) \vert \hat{\delta}_2(s_2^*) & s \in S_1 \\ \hat{\delta}2(s) & s \in S_2 \end{cases}$
$e_1^{(b)}$	$\{s^*\} + S_1$	s^*	$\begin{cases} \{\overline{b}\} & s = s^* \\ \langle\{\overline{b}\} \vert \hat{\epsilon}1(s) & s \in S_1 \end{cases}$	$\begin{cases} \langle\{b\} \vert \hat{\delta}_1(s_1^*) & s = s^* \\ \hat{\delta}_1(s) \cup \langle\{b\} \vert \langle\hat{\epsilon}1(s) \vert \hat{\delta}_1(s_1^*) & s \in S_1 \end{cases}$

TABLE I: Symbolic Thompson's Construction

and S_2, and for any two transition dynamics $\delta_1(s_1) : \mathcal{P}(\mathsf{Bool} \times S_1 \times K)$ and $\delta_2(s_2) : \mathcal{P}(\mathsf{Bool} \times S_2 \times K)$, then $\delta_1^*(s_1) + \delta_2^*(s_2) : \mathcal{P}(\mathsf{Bool} \times (S_1 + S_2) \times K)$ is the bifunctorial lift via $+$.

One notable difference between the original construction [9] 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 slightly easier. However, in Section VII-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, how does the number of reachable state relate to the size of the expression, so that we can estimate the complexity of the equivalence checking algorithm. Turns out all of these questions can be answered by a connection by a homomorphism from symbolic Thompson's construction to the symbolic derivatives.

Theorem 21. *Given any GKAT expression e, the resulting symbolic GKAT coalgebra from Thompson's construction $\hat{S}_e$ have a homomorphism to derivatives $h : \hat{S}_e \to \hat{D}$, s.t. for the start state $s^* \in S, h(s^*) = e$.*

Proof. By induction on the structure of e. We will recall that $h : \hat{S}_e \to \langle e \rangle_D$ 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 $h_1 : \hat{S}_{e_1} \to \langle e_1 \rangle_D$ and $h_2 : \hat{S}_{e_2} \to \langle e_2 \rangle_D$. 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 $s \in \hat{S}_{e_1} \cap \hat{S}_e$, we have

$$s \Rightarrow_{\hat{S}_e} c \text{ iff } s \Rightarrow_{\hat{S}_{e_1}} c \text{ iff } h_1(s) \Rightarrow_{\hat{D}} c \text{ iff } h(s) \Rightarrow_{\hat{D}} c$$

$$s \xrightarrow{c|p}_{\hat{S}_e} s' \text{ iff } s \xrightarrow{c|p}_{\hat{S}_{e_1}} s' \text{ iff } h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s') \text{ iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s')$$

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^*:

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

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

When $e \triangleq e_1; e_2$, by induction hypothesis, we have two homomorphisms $h_1 : \hat{S}_{e_1} \to \hat{D}$ and $h_2 : \hat{S}_{e_2} \to \hat{D}$. 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 $s \in \hat{S}_{e_1}$:

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

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

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

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

$$s \xrightarrow{c|p}_{\hat{S}_e} s' \text{ iff } s \xrightarrow{c|p}_{\hat{S}_{e_2}} s' \text{ iff } h_2(s) \xrightarrow{c|p}_{\hat{D}} h_2(s') \text{ iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s').$$

When $e \triangleq {e_1}^{(b)}$, by induction hypothesis, we have a homomorphism $h_1 : \hat{S}_{e_1} \to \hat{D}$; 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:

$$(s^* \Rightarrow_{\hat{S}_e} c) \text{ iff } (s^* \Rightarrow_{\hat{S}_e} c \text{ and } c = \overline{b}) \text{ iff } ({e_1}^{(b)} \Rightarrow_{\hat{D}} c \text{ and } c = \overline{b}) \text{ iff } (h$$

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

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

$$\begin{aligned} & s \Rightarrow_{\hat{S}_e} c \\ \text{iff } & (\exists a, \overline{b} \wedge a = c \text{ and } s \Rightarrow_{\hat{S}_{e_1}} a) \\ \text{iff } & (\exists a, \overline{b} \wedge a = c \text{ and } h_1(s) \Rightarrow_{\hat{D}} a) \\ \text{iff } & h_1(s); e_1^{(b)} \Rightarrow_{\hat{D}} c \text{ iff } h(s) \Rightarrow_{\hat{D}} c \end{aligned}$$

$$\begin{aligned} & s \xrightarrow{c|p}_{\hat{S}_e} s' \\ \text{iff } & (s \xrightarrow{c|p}_{\hat{S}_{e_1}} s' \text{ or } \exists a_1, a_2, b \wedge a_1 \wedge a_2 = c \text{ and } s \Rightarrow_{\hat{S}_{e_1}} a_1 \text{ and } s_1^* \xrightarrow{a_2|p} \\ \text{iff } & (h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s') \text{ or } \exists a_1, a_2, b \wedge a_1 \wedge a_2 = c \text{ and } h_1(s) \Rightarrow_{\hat{D}} a_1 \text{ an} \\ \text{iff } & (h_1(s) \xrightarrow{c|p}_{\hat{D}} h_1(s') \text{ or } \exists a_1, a_2, b \wedge a_1 \wedge a_2 = c \text{ and } h_1(s) \Rightarrow_{\hat{D}} a_1 \text{ an} \\ \text{iff } & (h_1(s); e_1^{(b)} \xrightarrow{c|p}_{\hat{D}} h_1(s'); e_1^{(b)}) \text{ iff } h(s) \xrightarrow{c|p}_{\hat{D}} h(s') \quad \square \end{aligned}$$

Theorem 21 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 22 (Correctness)**.** *Given any expression e and its Thompson's coalgebra $\hat{S}_e$ with a start state $s^* \in \hat{S}_e$, then the semantics of the start state is equivalent to the semantics of e: $[\![s^*]\!]_{\hat{S}_e} = [\![e]\!]$.*

A not so obvious consequence of the homomorphism in Theorem 21, is the complexity of the algorithm based on derivatives. Our bisimulation algorithm (Algorithm 4) 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 21.

Corollary 23. *There exists a surjective homomorphism $h' : \langle s^* \rangle_{\hat{S}_e} \to \langle e \rangle_{\hat{D}}$. 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. $h'(s) \triangleq h(s)$. Then we need to show that h' is well-defined ans 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' is point-wise equal to h and is defined on $\langle s^* \rangle_{\hat{S}_e}$, the range of h' contains its codomain $\langle e \rangle_{\hat{D}}$, showing that h' is surjective. ☐

An important consequence of corollary 23 is that $\langle s^* \rangle_{\hat{S}_e}$ will have no less state than $\langle e \rangle_{\hat{D}}$. However, it is important to notice that this does not mean running bisimulation algorithm on

VII. Implementation

A. Optimization

B. Performance

VIII. 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] Filippo Bonchi et al. "A General Account of Coinduction Up-To". In: *Acta Informatica* 54.2 (Mar. 2017), pp. 127–190. ISSN: 0001-5903, 1432-0525. DOI: 10.1007/s00236-016-0271-4. (Visited on 01/22/2022).

[3] 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).

[4] 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 Ramaswamy 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).

[5] 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).

[6] 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).

[7] 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.

[8] 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).

[9] 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