One-The-Fly Decision Procedures For GKAT

Cheng Zhang Qiancheng Fu Hang Ji Ines Santacruz Marco Gaboardi

Abstract

TODO: change all the e_b to b.

1 Introduction

Notation: In this paper, we will use un-curried 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 $(-) \times (-)$.

2 Preliminary

2.1 Concepts in Universal Coalgebra

Our notion of normalized bisimulation is inherently coalgebraic, thus it is empirical for us to recall some notions and theorems in 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 $\delta_S : S \to F(S)$. We typically call F the *signature* of the coalgebra, S the states of the coalgebra, and δ_S the transition function of the coalgebra. We will sometimes use the states S to denote the coalgebra, when no ambiguity can arise.

A homomorphism between two F-coalgebra S and T is a map $h : S \to T$ that preserves the transition function; diagrammatical, the following diagram commutes:

$$\begin{array}{ccc} S & \xrightarrow{h} & T \\ {\scriptstyle \delta_S}\downarrow & & \downarrow{\scriptstyle \delta_T} \\ F(S) & \xrightarrow{F(h)} & F(T) \end{array}$$

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 $S' \sqsubseteq S$:

$$\begin{array}{ccc} S' & \xhookrightarrow{i} & S \\ {\scriptstyle \delta_{S'}}\downarrow & & \downarrow{\scriptstyle \delta_S} \\ F(S') & \xhookrightarrow{F(i)} & F(S) \end{array}$$

In fact, the transition function $\delta_{S'}$ is uniquely determined by the states S' [5, Proposition 6.1].

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

Lemma 1 (Theorem 6.3 [5]). *given a homomorphism $h : S \to T$, and sub-coalgebras $S' \sqsubseteq S$ and $T' \sqsubseteq T$, then*

$$h(S') \sqsubseteq T \text{ and } h^{-1}(T') \sqsubseteq S.$$

One particularly important sub-coalgebra of and coalgebra S is the least coalgebra generated by a single element s, we will typically denote it as $\langle s \rangle_S$, and call it *principle sub-coalgebra* generated by s. We will omit the subscript S when it can be inferred from context or irrelevant. As we will see later, principle sub-coalgebra $\langle s \rangle_S$ is important because it models all the reachable state of s in S.

Principle sub-coalgebra always exists, because sub-coalgebra of any coalgebra forms a complete lattice [5, theorem 6.4]. And 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 $h : S \to T$:*

$$h(\langle S \rangle_S) = \langle h(s) \rangle_T$$

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, $h(\langle S \rangle_S) \sqsubseteq T$.

Finally, take any $T' \sqsubseteq T$ and $h(s) \in T'$, recall that by lemma 1, $h^{-1}(T') \sqsubseteq S$. We can then derive that $h(\langle s \rangle_S) \sqsubseteq T'$:

$$\begin{aligned} h(s) \in T' &\implies s \in h^{-1}(T') \\ &\implies \langle s \rangle_S \sqsubseteq h^{-1}(T') && \text{definition of } \langle s \rangle_S \\ &\implies h(\langle s \rangle_S) \sqsubseteq T' && \text{take the image of } h \text{ and lemma 1} \end{aligned}$$

Hence $h(\langle s \rangle_S)$ is the smallest sub-GKAT coalgebra of T 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 T, the *behavioral equivalence* between states in S and T can be computed by a notion called *bisimulation*. A relation $\sim \subseteq S \times T$ is called a *bisimulation relation* if it forms a F-coalgebra:

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

And its projection functions $\pi_1 : S \times T \to S$ and $\pi_2 : S \times T \to T$ are both homomorphisms:

$$\begin{array}{ccccc} S & \xleftarrow{\pi_1} & \sim & \xrightarrow{\pi_2} & T \\ \downarrow{\scriptstyle \delta_S} & & \downarrow{\scriptstyle \delta_\sim} & & \downarrow{\scriptstyle \delta_T} \\ F(S) & \xleftarrow[F(\pi_1)]{} & F(\sim) & \xrightarrow[F(\pi_2)]{} & F(T) \end{array}$$

As we will see later, bisimulation or variation of bisimulation indeed correspond to important semantical equivalences.

2.2 Guarded Kleene Algebra With Tests

2.3 GKAT Coalgebra

GKAT coalgebra [7, 6], is a coalgebraic systems for GKAT. Specifically GKAT coalgebras over a alphabet K, B are coalgebras over the following functor:

$$G(S) \triangleq (2 + S \times K)^{\mathbf{At}_B},$$

where $2 \triangleq \{\text{acc}, \text{rej}\}$. 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 $\delta(s, \alpha) = \text{rej}$; accept α, denoted $\delta(s, \alpha) = \text{acc}$; 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 [3], 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 further optimized symbolic algorithm than KCT [4], but also present challenges. Specifically, GKAT coalgebra requires normalization to compute finite trace equivalences [7], where we will remove all the state that cannot lead to acceptance. It seems like we need to traverse the entire automaton to identify these "dead states", however we have shown that these dead state detection can be invoked lazily, only when discrepancy between bisimulation are found.

2.4 Liveness and Sub-GKAT coalgebras

Traditionally, live and dead states are defined by whether they can reach an accepting state [7]. However, recall that principle sub-coalgebra $\langle s \rangle_S$ models 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. $\delta(s, \alpha) = \text{acc}$. 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 3. *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 4 (homomorphism perserves liveness)**.** *Given a homomorphism $h : S \to T$ 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

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

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

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

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 $h(\langle s \rangle_S) = \langle h(s) \rangle_T$ contains an accepting state. Therefore s is live in S if and only if $h(s)$ is live in T. □

Corollary 5 (sub-coalgebra perserves liveness)**.** *Given a sub-coalgebra $S' \sqsubseteq S$, then for all states $s \in S'$,*

$$s \text{ is live in } S' \iff s \text{ is live in } S.$$

Proof. take the homomorphism h in theorem 4 to be the inclusion homomorphism $i : S' \to S$. □

Corollary 6 (bisimulation preserves liveness)**.** *If there exists a bisimulation $\sim$ between GKAT coalgebra S and T s.t. $s \sim t$ for some states $s \in S$ and $t \in T$, then s and t 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 $\pi_2 : \sim \to T$ are homomorphisms. Therefore,

$$\begin{aligned} s \text{ is live in } S &\iff \pi_1((s,t)) \text{ is live in } S && \pi_1((s,t)) = s \\ &\iff (s,t) \text{ is live in } \sim && \pi_1 \text{ is a homomorphism} \\ &\iff \pi_2((s,t)) \text{ is live in } T && \pi_2 \text{ is a homomorphism} \\ &\iff t \text{ is live in } T && \pi_2((s,t)) = t \end{aligned}$$

□

2.5 Normalization And Trace Semantics

(Possibly infinite) trace model $\mathcal{G}_\omega$ is the natural semantics of states in a GKAT coalgebra, specifically it forms the final coalgebra of GKAT coalgebras [6]. The finality of the model means 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 semantical equivalences can indeed be identified by bisimulation [6]:

$$[\![s]\!]^\omega_S = [\![t]\!]^\omega_T \iff \text{exists a bisimulation } \sim \subseteq S \times T, \text{ s.t. } s \sim t.$$

The infinite trace equivalences is relatively easy to compute efficiently, as bisimulation is, in general, compatible with derivative based computation on-the-fly algorithm [3, 1, 4]. However, the *finite* trace model $\mathcal{G}$ is only the final coalgebra of GKAT coalgebras without dead states, which we call *normal GKAT coalgebra* [7]. Fortunately every GKAT coalgebra can be normalized by rerouting all the transition from dead states to rejection

$$\begin{aligned} \mathrm{norm}(\delta_S) &: S \to G(S) \\ \mathrm{norm}(\delta_S)(s,\alpha) &\triangleq \begin{cases} \mathrm{rej} & \text{if } \delta_S(s,\alpha) = (s', p) \text{ and } s' \text{ is dead} \\ \delta_S(s,\alpha) & \text{otherwise} \end{cases} \end{aligned}$$

We denote the normalized coalgebra $(S, \mathrm{norm}(\delta_S))$ as $\mathrm{norm}(S)$. This means that the finite trace semantics $[\![-]\!]$ is the unique coalgebra homomorphism $\mathrm{norm}(S) \to \mathrm{norm}(\mathcal{G})$. Hence the finite trace equivalence between $s \in S$ and $t \in T$ can be computed by first normalizing S and T, then decide whether there is a bisimulation on $\mathrm{norm}(S)$ and $\mathrm{norm}(T)$ that includes (s, t). For a more intuitive account for the trace semantics, we refer the reader to the work of Smolka et al. [7].

Then by the finality of GKAT colagebra, the soundness and completeness of bisimulation is a simple corollary in universal coalgebra [7, 2, 5]. No matter through explicit construction [7] or (weak) pullback [2, 5], both completeness proof shows that the language equivalence:

$$\equiv \triangleq \{(s,t) \mid [\![s]\!]_S = [\![t]\!]_T\}$$

is indeed a bisimulation. This result, together with soundness result $s \sim t \implies [\![s]\!]_S = [\![t]\!]_T$ means that the language equivalence $\equiv$ is indeed the largest bisimulation, which allows us to work with bisimulation equivalence instead of bisimulation.

Definition 2. *A* bisimulation equivalence *in S is a bisimulation between S and itself, and it is also an equivalence relation.*

Theorem 7. *Given two states $s, t \in S$, then there exists a bisimulation $\sim$ s.t. $s \sim t$ if and only if there exists a bisimulation equivalence $\simeq$ s.t. $s \simeq t$.*

Proof. The $\implies$ direction can just take $\simeq$ to be the language equivalence $\equiv$, which is a bisimulation equivalence, and because $\equiv$ is maximal, therefore $\sim \sqsubseteq \equiv$, and $(s,t) \in \sim \subseteq \equiv$.

The $\impliedby$ direction is true because all bisimulation is a bisimulation equivalence, thus we can take $\sim$ to just be the given bisimulation equivalence $\simeq$. □

3 On-The-Fly Bisimulation

The original algorithm for deciding GKAT equivalences [7] 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 [7], 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.

Theorem 8 (sub-coalgebra perserve 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

$$\begin{array}{ccccccccc} S & \xleftarrow{i} & S' & \xleftarrow{\pi_1} & \sim & \xrightarrow{\pi_2} & T' & \xhookrightarrow{i} & T \\ \downarrow{\scriptstyle \delta_S} & & \downarrow{\scriptstyle \delta_S} & & \downarrow{\scriptstyle \delta_\sim} & & \downarrow{\scriptstyle \delta_S} & & \downarrow{\scriptstyle \delta_T} \\ G(S) & \xleftarrow{G(i)} & G(S') & \xleftarrow{G(\pi_1)} & G(\sim) & \xrightarrow{G(\pi_2)} & T' & \xhookrightarrow{G(i)} & T \end{array}$$

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

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

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:

$$\begin{array}{ccc} \sim_{S',T'} & \xrightarrow{i} & \equiv_{S',T'} \\ {\scriptstyle i}\downarrow & \lrcorner & \downarrow{\scriptstyle i} \\ \sim & \xrightarrow[i]{} & \equiv_{S,T} \end{array}$$

Recall that elementary polynomial functor [2] like G preserves pullback, hence the pullback also uniquely generates a GKAT coalgebra [5] □

Lemma 9 (bisimulation between dead states)**.** *Given two dead states $s \in S$ and $t \in T$, then the singleton bisimulation*

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

is a bisimulation between S and T.

Proof. By computation □

Theorem 10 (inductive construction)**.** *Given two GKAT coalgebra S and T, and two of their elements $s \in S$ and $t \in T$, there exists a bisimulation $\sim \subseteq \langle s \rangle \times \langle t \rangle$ s.t. $s \sim t$, if and only if all of the following holds:*

1. *for all $\alpha \in \mathbf{At}$, $\delta_S(s,\alpha) = \mathrm{acc} \iff \delta_T(t,\alpha) = \mathrm{acc}$;*
2. *If $\delta_S(s,\alpha) = (s',p)$ and $\delta_T(t,\alpha) = (t',p)$, then there exists a bisimulation $\sim_{s',t'}$ on $\langle s' \rangle$ and $\langle t' \rangle$, s.t. $s' \sim_{s',t'} t'$;*

3. *If $\delta_S(s,\alpha) = (s', p)$ and $\delta_T(t,\alpha) = (t', q)$, s.t. $p \neq q$, then both s' and t' are dead;*

4. *s reject α or transition to a dead state via α if and only if t rejects α or transition to a dead state via α.*

Proof. We first prove $\Longrightarrow$ direction, recall the definition of bisimulation:

$$\begin{array}{ccccc} S & \xleftarrow{\pi_1} & \sim & \xrightarrow{\pi_2} & T \\ {\scriptstyle \mathrm{norm}(\delta_S)}\downarrow & & \downarrow{\scriptstyle \delta_\sim} & & \downarrow{\scriptstyle \mathrm{norm}(\delta_T)} \\ G(S) & \xleftarrow[G(\pi_1)]{} & G(\sim) & \xrightarrow[G(\pi_2)]{} & G(T) \end{array}$$

The condition 1 holds:

$$\begin{aligned} \delta_S(s,\alpha) = \mathrm{acc} &\Longleftrightarrow \mathrm{norm}(\delta_S)(s,\alpha) = \mathrm{acc} \Longleftrightarrow \mathrm{norm}(\delta_\sim)((s,t),\alpha) = \mathrm{acc} \\ &\Longleftrightarrow \mathrm{norm}(\delta_T)(t,\alpha) = \mathrm{acc} \Longleftrightarrow \delta_T(t,\alpha) = \mathrm{acc} \end{aligned}$$

The condition 2 holds, by case analysis on the liveness of s' and t'. First note that s' and t' 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 $\mathrm{norm}(\delta_T)(t',\alpha)$. Finally because $s \sim t$, then

$$s' \text{ is live} \Longleftrightarrow \mathrm{norm}(\delta_S)(s,\alpha) = (s', p) \Longleftrightarrow \mathrm{norm}(\delta_T)(t,\alpha) = (t', p) \Longleftrightarrow t' \text{ is live.}$$

- If both s' and t' are live, then $s' \sim t'$, the bisimulation $\sim_{s',t'}$ is just $\sim$ restricted to $\langle s' \rangle$ and $\langle t' \rangle$.
- If both s' and t' are dead, then $\sim_{s',t'}$ can just be the singleton relation, according to lemma 9.

The condition 3 holds: by the proof of condition 2, s' and t' 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.

The condition 4 holds:

$$\begin{aligned} \delta_S(s,\alpha) \text{ rejects or transition to dead states} &\Longleftrightarrow \mathrm{norm}(\delta_S)(s,\alpha) = \mathrm{rej} \\ &\Longleftrightarrow \mathrm{norm}(\delta_T)(t,\alpha) = \mathrm{rej} \\ &\Longleftrightarrow \delta_T(t,\alpha) \text{ rejects or transition to dead states.} \end{aligned}$$

We then show the $\Longleftarrow$ direction, we use $\equiv_{s',t'}$ to denote the maximal bisimulation between $\langle s' \rangle$ and $\langle t' \rangle$.

$$\sim' \triangleq \bigcup \{ \equiv_{s',t'} \mid \exists \alpha \in \mathbf{At}, p \in K, \delta_S(s,\alpha) = (s', p) \text{ and } \delta_T(t,\alpha) = (t', p) \}.$$

Notice because $\langle s' \rangle \sqsubseteq \langle s \rangle$ and $\langle t' \rangle \sqsubseteq \langle t \rangle$, then $\equiv_{s',t'}$ is a bisimulation between $\langle s \rangle$ and $\langle t \rangle$, and because bisimulation is closed under arbitrary union [5], then $\sim'$ is a bisimulation between $\langle s \rangle$ and $\langle t \rangle$.

We then augment $\sim'$ with (s,t) to obtain the bisimulation we required:

$$\sim \triangleq \sim' \cup \{(s,t)\} \qquad \delta_\sim((s_1,t_1),\alpha) \triangleq \begin{cases} \delta_{\sim'}((s,t),\alpha) & (s,t) \neq (s_1,t_1) \\ \mathrm{acc} & \mathrm{norm}(\delta_S)(s,\alpha) = \mathrm{norm}(\delta_T)(s,\alpha) = \mathrm{acc} \\ \mathrm{rej} & \mathrm{norm}(\delta_S)(s,\alpha) = \mathrm{norm}(\delta_T)(s,\alpha) = \mathrm{rej} \\ ((s_2,t_2),p) & \mathrm{norm}(\delta_S)(s,\alpha) = (s_2,p) \text{ and } \mathrm{norm}(\delta_T)(s,\alpha) = (t_2,p) \end{cases}$$

The above definition of $\delta_\sim$ is indeed well-defined, by case analysis on the result of δ_S and δ_T using the condition above:

- If $\delta_S(s,\alpha) = \mathrm{acc}$, then by condition 1, $\delta_T(t,\alpha) = \mathrm{acc}$ therefore

$$\mathrm{norm}(\delta_S)(s,\alpha) = \mathrm{norm}(\delta_T)(s,\alpha) = \mathrm{acc}\,.$$

- If $\delta_S(s,\alpha)$ transitions to a dead state or reject, then by conditions 4 $\delta_T(t,\alpha)$ will also transition to a dead state or reject, then
$$norm(\delta_S)(s,\alpha) = \text{norm}(\delta_T)(s,\alpha) = \text{rej}\,.$$
- If $\delta_S(s,\alpha) = (s',p)$, then by condition 1 and condition 4, $\delta_T(t,\alpha) = (t',q)$. By the contrapositive of condition 3, if either s,t are live, then $p = q$.

  Then by condition 2, there exists a bisimulation $\sim_{s',t'}$ between $\langle s'\rangle$ and $\langle t'\rangle$ s.t. $s' \sim_{s',t'} t'$. Because bisimulation preserves liveness (corollary 6), s',t' has to be both dead or live, the both dead case is handled by the previous item, both live case will give us the case we desired:
$$\text{norm}(\delta_S)(s,\alpha) = (s',p) \text{ and } \text{norm}(\delta_T)(t,\alpha) = (t',p)$$

And the diagram bisimulation needing to satisfy can be verified by unfolding the definition. □

The above theorem already gives us a way to recursively construct a algorithm that include $s \sim t$, this consequently will let us decide the trace equivalence of s and t: $[\![s]\!] = [\![t]\!]$. 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 10.

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

Theorem 11. *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 6, then t is dead.

And if both t and s is dead, then a bisimulation can by constructed by lemma 9. □

4 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 10), we will need to determine the liveness of the state. This can be simply computed via a DFS from the state being checked.

Algorithm 1 Check whether a state s is dead

```
function isDeadLoop(s ∈ S, explored)
    if s ∈ explored then return explored
    else
        for α ∈ At do
            match δ_S(s, α) with
                case acc then return none                          ▷ s transition to accept
                case (s', p) then
                    if IsDead(s') = none then return none          ▷ s transitions to a live state s'
                    else explored ← (explored ∪ isDeadLoop(s', explored))
    return explored
```

By lemma 3, 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.

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

$$\begin{aligned}
&\exists \text{ bisimulation } \sim \subseteq \langle s\rangle \times \langle t\rangle \text{ s.t. } s \sim t \\
&\iff \exists \text{ bisimulation } \sim \subseteq (\langle s\rangle \cup \langle t\rangle) \times (\langle s\rangle \cup \langle t\rangle) \text{ s.t. } s \sim t && \text{theorem 8} \\
&\iff \exists \text{ bisimulation equivalence } \simeq \subseteq (\langle s\rangle \cup \langle t\rangle) \times (\langle s\rangle \cup \langle t\rangle) \text{ s.t. } s \simeq t && \text{theorem 7}
\end{aligned}$$

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

```
deadStates ← ∅
function isDead(s ∈ S)
    if s ∈ deadStates then return true
    else if isDeadLoop(s, ∅) = none then return false
    else
        deadStates ← (deadStates ∪ isDeadLoop(s, ∅))
        return true
```

we can safely replace the bisimulation in inductive construction (theorem 10) 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 10; except by theorem 11, in the special case where s or t is dead, then we will only need to check whether the other 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 then return isDead(t)
    else if t ∈ deadStates then return isDead(s)
    else
        for α ∈ At do                                  ▷ Inductive construction, theorem 10
            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
```

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 10 and 11.

Remark 3. *The caching of dead state and the shortcut 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.*

5 Symbolic Algorithm

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

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

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

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

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 4 (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 $\bot$. 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.

We can then migrate the normalized bisimulation algorithm to the symbolic setting, we will first prove an inductive construction theorem like theorem 10.

Theorem 12 (Symbolic Inductive Construction)**.** *Given two symbolic GKAT coalgebra $\widehat{S} = \langle S, \hat{\epsilon}_S, \hat{\delta}_S \rangle$ and $\widehat{T} = \langle T, \hat{\epsilon}_T, \hat{\delta}_T \rangle$ 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:*

- *$\hat{\epsilon}_S(s) \equiv \hat{\epsilon}_T(t)$;*
- *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;*
- *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 10 i.e. all the above condition holds if and only if all the condition in theorem 10 holds in the lowered coalgebra. □

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 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).

[4] 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).

[5] 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.

[6] 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).

[7] 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.

A diagrammatical characterization of normalized homomorphism