A Symbolic Decision Procedure for GKAT

Cheng Zhang

Abstract

1 Symbolic Derivatives

Given a finite set K, a *symbolic Guarded Kleene Coalgebra with Tests* (sGKCT) $\mathcal{S} \triangleq \langle S, \epsilon, \delta\rangle$ over a boolean algebra $\mathcal{B}$ consists of a state set S consists of a accepting boolean ϵ and a transition function δ:

$$\epsilon : S \to \mathcal{B}, \qquad \delta : S \to S \to K \to \mathcal{B},$$

where for all states $s \in S$, all the booleans are "disjoint"; namely the conjunction of any two expression from the set $\epsilon(s) + \{\delta(s, s', p) \mid s' \in S, p \in K\}$ are false. The ordering on $\mathcal{S}$ is defined pointwise, namely

$$\langle \epsilon_1, \delta_1\rangle \leq \langle \epsilon_2, \delta_2\rangle \text{ when } \forall s, s' \in S, \forall p \in K, \epsilon_1(s) \leq \epsilon_2(s) \text{ and } \delta_1(s, s', p) \leq \delta_2(s, s', p),$$

where the ordering on the right hand side is defined in the boolean algebra.

Intuitively, the elements of boolean is treated as the "symbolic" transitions, which can be think of as a set of labels by the classical stone's representation theorem. In particular, when the boolean algebra is the free boolean algebra BExp_B generated by a finite set B, the labels can be considered as atoms, since $\mathsf{BExp}_B \cong 2^{\mathbf{At}_B}$. Then we can explain the intuition for the definition of GKCT

- $\epsilon(s)$ can be thought of as a set of labels that is accepted by the state s.
- $\delta(s, s', p)$ denotes all the labels that can transition from s to s' while executing p.

Then the labels that is not in either operations are implicitly rejecting.

Given a sGKCT $\mathcal{S} \triangleq \langle S, \epsilon, \delta\rangle$ and a state $s \in S$, we use the notation $s \Rightarrow b$ to denote that $\epsilon(s) \geq b$, and we use the notation $s \xrightarrow{b|p} s'$ to denote that $\delta(s)_3(s', p) \geq b$, where the ordering is the typical ordering in boolean algebra.

The symbolic derivatives for GKAT forms a GKCT where the states are represented by GKAT expression $\mathsf{GKAT}_{K,B}$, and the boolean algebra is the free boolean algebra BExp_B over the test alphabet B:

$$\epsilon : \mathsf{GKAT}_{K,B} \to \mathsf{BExp}_B; \qquad \delta : \mathsf{GKAT}_{K,B} \to \mathsf{GKAT}_{K,B} \to K \to \mathsf{BExp}_B.$$

the accepting map ϵ and transition map δ can be defined by the smallest maps that satisfy the following rules:

$$\frac{}{b \Rightarrow b} \qquad \frac{}{p \xrightarrow{1|p} 1} \qquad \frac{e \Rightarrow a}{e +_b f \Rightarrow ba} \qquad \frac{f \Rightarrow a}{e +_b f \Rightarrow \bar{b}a} \qquad \frac{e \xrightarrow{a|p} e'}{e +_b f \xrightarrow{b\wedge a|p} e'} \qquad \frac{f \xrightarrow{a|p} f'}{e +_b f \xrightarrow{\bar{b}\wedge a|p} f'}$$

$$\frac{e \Rightarrow a \qquad f \Rightarrow b}{e \cdot f \Rightarrow a \wedge b} \qquad \frac{e \Rightarrow a \qquad f \xrightarrow{b|p} f'}{e \cdot f \xrightarrow{a\wedge b|p} f'} \qquad \frac{e \xrightarrow{b|p} e'}{e \cdot f \xrightarrow{b|p} e' \cdot f} \qquad \frac{}{e^{(b)} \Rightarrow \bar{b}} \qquad \frac{e \xrightarrow{a|p} e'}{e^{(b)} \xrightarrow{b\wedge a|p} e' \cdot e^{(b)}}$$

2 Implementing derivatives

Since the rules listed above is monotonic, that is if we enlarge the boolean in the premise, we also enlarge the boolean in the result. So we can simply pick the largest boolean on the premise, which is either $\epsilon(e)$ or $\delta(e, e', p)$. Since the disjunction of two boolean $a \vee b$ is the smallest boolean that is greater than both a and b, thus we can implement ϵ and δ by iteration through all the rules, and take the disjunction of all the possible booleans.

$$\begin{aligned} \epsilon(b) &\triangleq b & \epsilon(e +_b f) &\triangleq (b \wedge \epsilon(e)) \vee (\bar{b} \wedge \epsilon(f)) \\ \epsilon(q) &\triangleq 0 & \epsilon(e \cdot f) &\triangleq \epsilon(e) \wedge \epsilon(f) \\ \epsilon(e^{(b)}) &\triangleq \bar{b} \end{aligned}$$

However, the derivative function

$$\delta : \mathsf{GKAT}_{K,B} \to \mathsf{GKAT}_{K,B} \to K \to \mathsf{BExp}_B,$$

poses challenges in the implementation, since $\mathsf{GKAT}_{K,B}$ is infinite, so it will be impractical to search through all the expressions. However, we can treat the function δ extensionally:

$$\mathsf{GKAT}_{K,B} \to K \to \mathsf{BExp}_B \subseteq 2^{\mathsf{GKAT}_{K,B} \times K \times \mathsf{BExp}_B},$$

and then because the boolean expression don't overlap for each input, hence the boolean expression are necessarily unequal; we can model the set by a partial map, thus all the δ can be represented as a function to the following partial maps.

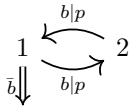
$$\begin{aligned} \delta &: \mathsf{GKAT}_{K,B} \to (\mathsf{BExp}_B \nrightarrow \mathsf{GKAT}_{K,B} \times K) \\ \delta(b) &\triangleq \{\} \\ \delta(q) &\triangleq \{1 \mapsto (1, p)\} \\ \delta(e +_b f) &\triangleq \{b \wedge a \mapsto (e', p) \mid a \mapsto (e', p) \in \delta(e)\} \cup \{\bar{b} \wedge a \mapsto (f', p) \mid a \mapsto (f', p) \in \delta(f)\} \\ \delta(e \cdot f) &\triangleq \{b \mapsto (e' \cdot f, p) \mid b \mapsto (e, p) \in \delta(e)\} \cup \{\epsilon(e) \wedge b \mapsto (f', p) \mid b \mapsto (f', p) \in \delta(f)\} \\ \delta(e^{(b)}) &\triangleq \{b \wedge a \mapsto (e' \cdot e^{(b)}, p) \mid a \mapsto (e', p) \in \delta(e)\}. \end{aligned}$$

FIXME: we still need to think about what representation do we want to go with, with this representation, you can also have duplicated K. We need a good example here of why don't we go with map. One good argument is that this representation is complex.

3 On the Fly Normalization

Normalization of a GKAT automaton will remove all the dead state, which are states that cannot lead to an accepting transitions. However, this process is hard to do on-the-fly, mainly because of the difficulty of handling infinite loops.

Example 1. *We can consider a simple extension to a BFS algorithm, where we will mark a state as either live, dead, or searching. A state is marked as "searching" when it has been explored but not yet known to be dead or alive yet, and when a state loops back to a searching state, then we can declare the detection of a infinite loop, hence marking the transition dead. We consider the following coalgebra over a two element set* $\{1, 2\}$*:*



It is clear that both states 1 *and* 2 *are live, since* 1 *is accepting and* 2 *have a transition to* 1*. However, if we run our search algorithm: to determine whether* 1 *is live, we will need to compute the backtrack of both of its transitions. We will compute the backtrack from b first, which computes the liveness of* 2*. Thus we will mark* 1 *as searching. Yet state* 2 *only have one transition to* 1*, because* 1 *is marked as searching, the algorithm will think a infinite loop is discovered and mark* 2 *as dead, which doesn't align with the reality.*

Thus, it is crucial to resolve these loops when we a designing a on-the-fly algorithm. In fact, we can utilize a common graph theory technique called *graph condensation* to remove the loops, where it treat all thee strong connected component as a vertex as a new graph, and the edges between the components are preserved. The resulting graph is a directed acyclic graph (DAG), then we can utilize a simple search to find the live state.

Definition 1 (Graph For sGKCT)**.** *Given a sGKCT* $\mathcal{S} \triangleq \langle S, \delta, \epsilon \rangle$*, we can construct a (potentially infinite) graph, where the vertices are* S *and the edge* (s, s') *is in the graph when connected there exists a* $p \in K$*, s.t.* $\delta(s, s', p) \neq 0$

We will also recall the definition of strongly connected components

Definition 2 (Walk and strongly connected components)**.** *A walk from vertex* s_1 *to vertex* s_2 *in a graph is a sequence of vertices* $[s_1, s_2, ..., s_n]$*, s.t. for all* i*,* (s_i, s_{i+1}) *are edges in the graph. A strongly connected component is a equivalence class on the vertices, where two vertices* s_1 *and* s_2 *are equivalent when there is a walk from* s_1 *to* s_2 *and there is a walk from* s_2 *to* s_1*.*

And in fact strongly connected components has strong connections with liveness of states.

Lemma 1. *By induction on the length of the walk, if there is a walk from state* s_1 *and* s_2 *then if* s_2 *is live and* s_1 *is live. Thus by definition of strongly connected component, a strongly connected component is live if and only if any of the state in the component is live. And consequently, all the states in a strongly connected component are either all dead or all live.*

To identify the strongly connected components, we can run the Tarjan's algorithm along side the normal Hopcroft and Karp's bisimulation algorithm. The Tarjan's algorithm is a one pass DFS based algorithm where it assigns each state a *low-link value*, at the end of the algorithm, all the algorithm with the same low-link value will be in the same strongly connected components. Thus instead of marking each state with dead or live, we will mark each state with a low-link value, and each low-link value is marked with either dead or alive. This kind of mappings are enabled by the fact that all states in a strongly connected components are either all dead or all live.

4 The Algorithm

FIXME: proof a theorem that if the transformation shares boolean with the original program, then the decision procedure is nearly linear time.