LEARNING SYMBOLIC AUTOMATA FROM SYMBOLIC COUNTER-EXAMPLES

BENJAMIN CAULFIELD, LORIS D'ANTONI, AND SANJIT SESHIA

In the work by D'Antoni & Drews, each new counter-example is added to the observation table, which affects how the hypothesis automaton is built. However, it is not clear how to incorporate new information when symbolic counter-examples are given. We modify the algorithm from D'Antoni & Drews to account for the *observed partition*, as described in the below definition. In this new algorithm, which we call Λ^*_{sym}, the observed partition is used in finding the separating predicates when the hypothesis automaton is created. It is updated when a new counter-example is given and when a new state is added.

Definition 3.1. (Observed Partition) During the execution of Λ^*_{sym}, the set S contains strings representing the set of observed states. Te *observed partition* of a state $s \in S$, written $\mathcal{P}(s)$, is a pair (U, ϕ_{neg}) where $U \subseteq 2^{\Phi \times S}$ with the following properties:

- There is an $s \in S$ such that $(\phi_{neg}, s) \in U$
- $[\![\phi_{neg}]\!] = [\![\neg[\bigvee_{\phi \in \mathcal{P}_s(S)} \phi]]\!]$
- For all distinct $(\phi', s'), (\phi'', s'') \in U$, $[\![\phi']\!] \cap [\![\phi'']\!] = \emptyset$

We define $\mathcal{P}_{s'}(s) := \{\phi \mid (\phi, s') \in \mathcal{P}(s)\}$ and $\mathcal{P}_S(s) := \bigcup_{s' \in S} \mathcal{P}_{s'}(s)$. The observed partition of S (hearafter referred to as the *observed partition*) is the function $\mathcal{P} : S \to 2^{\Phi \times S}$ mapping each $s \in S$ to the observed partition of s.

3.1. **Building the Symbolic Automaton.** The process for building the symbolic automaton from an observation table and observed partition is the same as in Λ^*, with the exception of the formation of transitions. In Λ^*, the transitions are guarded by predicates formed from a partitioning function, P, which is based solely on the evidence automaton. In addition to the evidence automaton, Λ^*_{sym} will use the observed partition to form a *symbolic partitioning function*, as described in the following definition.

Definition 3.2. (Symbolic Partitioning Function) A symbolic partitioning function for a Boolean algebra $\mathcal{A} = (\mathcal{D}, \Psi, [\![\_]\!], \bot, \top, \vee, \wedge, \neg)$ is a function $P_{sym} : (2^{\mathcal{D}}) \times 2^{\Phi \times S} \to \Psi$. It takes as input a list $L_D := l_1 ... l_k$ of disjoint sets of elements in $\mathcal{D}$ and the observed

definitions from Drews & D'Antoni (and def of A_{init}) will be added to the background info

is this the phrase we want to use?

Show graph of new framework?

Is this the right way to formalize this?

is there a better value that U to put here?

I'm trying to keep it in the same form as the partitioning function from the last paper. Do we need to include L_D in this definition? All of the partitioning functions we actually use are just based on the observed

partition $\mathcal{P}(s)$ for some state s (it doesn't need to know s). It returns a list $L_\Psi = \phi_1, \ldots, \phi_k$ of predicates in Ψ with the same properties as a Partitioning Function.

Partitioning function will be defined in background info.

Without making any assumptions on the oracle, it is not clear that much interesting information is stored in the observed partition. Different assumptions may require different P_{sym} functions.

However, there is a good partition function $\hat{P}$ that will yield nice learnability results when certain assumptions are made on the oracle. This partitioning function $\hat{P}$ takes in a list L_D (which it will ignore) and the observed partition (U, ϕ_{neg}) for a state s and returns a list $L_\Psi = \phi_1, \ldots, \phi_k$ such that $\phi_i := \bigvee_{(\phi, s_i) \in U} \phi$ for each i.

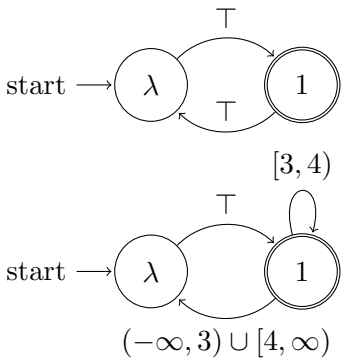
3.2. **Updating the Observed Partition.** When a symbolic automaton $M := (\mathcal{A}, Q, q_{init}, F, \Delta)$ is given to the oracle, a symbolic counter-example $\vec{\phi} := \phi_1, \phi_2, \ldots \phi_k$ is returned. The algorithm must then use $\vec{\phi}$ and the observation table T to update the observed partition $\mathcal{P}$.

To do this, the algorithm chooses a string $w := a_1, \ldots, a_k \in [\![\vec{\phi}]\!]$. For each i between 1 and k, the prefix $w_i := a_1, \ldots, a_i$ is added to the observation table and the state $s_i \in S$ such that $row(w_i) = row(s_i)$ is found. Let $\mathcal{P}(s_i) := (U_i, \phi_{neg})$. For each $(\phi, s) \in U_i$ such that $[\![\phi]\!] \cap [\![\phi_i]\!] \neq \emptyset$, (ϕ, s) is removed from U_i. For some $a \in [\![\phi]\!] \cap [\![\phi_i]\!]$, the set s' is found such that $row(s_i \cdot a) = row(s')$ and the pair $(\phi \wedge \phi_i, s')$ is added to U_i. The same is done for $\phi \wedge \neg\phi_i$, assuming $[\![\phi \wedge \neg\phi_i]\!] \neq \emptyset$. If $[\![\phi_i]\!] \cap [\![\phi_{neg}]\!] \neq \emptyset$, then ϕ_{neg} is set to $\phi_{neg} \wedge \neg\phi_i$.

is there an intuitive explanation of what this algorithm is doing?

Example 3.1. *Assume that Λ^*_{sym} is learning an automaton defined over $\mathcal{A}_{init}$ and has formed some observation table T with states $S := \{\lambda, 1\}$. The observed partition so far yields $\mathcal{P}(\lambda) := (\{(\top, 1)\}, \top)$ and $\mathcal{P}(1) := (\{(\top, \lambda)\}, \top)$. Using the partition function $\hat{P}$, the hypothesis M_1, shown in figure When M_1 is given to the oracle, the counter-example $\vec{\phi} := [0, 2), [3, 4)$ is given. This is used to update $\mathcal{P}$, so that $\mathcal{P}(\lambda) := (\{([0, 2), 1), ((-\infty, 0) \cup [2, \infty), 1)\}, (-\infty, 0) \cup [2, \infty))$ and $\mathcal{P}(1) := (\{((-\infty, 3) \cup [4, \infty), \lambda), ([3, 4), 1)\}, (-\infty, 3) \cup [4, \infty))$. The algorithm then creates the hypothesis M_2 shown in figure*

any way to make this more readable?



I need to make font smaller, remember that top edge is actually $[0, 2) \cup (-\infty, 0) \cup [2, \infty)$

4. Angelic Learning

4.1. **Maximal Paths.**