Approximate learning of k-step invariant for probabilistic program

Arnab

September 24, 2024

Chapter 1

Approximate learning of k-step invariant

1.1

Definition 1.1.1. k-step invariant of a probabilistic program P: Given a probabilistic program P and a set S of initial states, the k-step invariant or the k-iterative closure of P with respect to S is given by the set $cl_k(S) = \{\sigma \in \{0,1\}^{|V|} \mid \sigma \text{ is reachable from } S \text{ in at most } k \text{ iterations of } P\}$. Note that $cl_k(S) = \cup_{i=0}^{k} \psi^i(S) = \{\sigma \in \{0,1\}^{|V|} \mid \sigma \text{ is reachable in at most } k \text{ steps from } S\}$, where ψ denotes the single-iteration input-output function for the program P.

Definition 1.1.2. Distance between a candidate and the k-step invariant: Given a probabilistic program P and a set S of initial states, the distance between a given candidate T and the k-step invariant $cl_k(S)$ is given by $d(T, cl_k(S)) = \sum_{y \in (cl_k(S) \setminus T)} \Pr_{x \sim_U S}[y \text{ is reachable from } x \text{ in at most } k \text{ iterations }] = \Pr_{x \sim_U S}[(cl_k(S) \setminus T) \text{ is reachable from } x \text{ in at most } k \text{ iterations }]$.

Assumption 1.1.1. Succint description of the true k-step invariant: We have assumed that the true k-step invariant for a given probabilistic program P with respect to a set S of initial states can be expressed in the form of a CNF of bounded size (polynomial) in the number of variables.

Assumption 1.1.2. Sampling access to internal random variables of P: We have assumed that the probabilistic program can be allowed to run in a deterministic manner by fixing a random seed, given sampling access to the internal random variables R.

Note. Monotonicity of candidates with respect to violating transitions: An important thing to note here is that for the ease of analysis, we are only interested in the family Γ of candidates such that given any candidate $T \in \Gamma$, any k-length transition starting from S does not return back to T once it goes out of it.

<u>Problem</u> **1.1.3. (Additive approximation of the distance of a candidate T from k-step invariant $cl_k(S)$):** Given a probabilistic program P, a set S of initial states , the number of iterations k and a candidate T, parameters $\epsilon, \delta \in (0,1]$ output an ϵ-additive approximation of the distance $d(T, cl_k(S))$ with probability at least $1-\delta$.

Theorem 1.1.4 (Correctness of `DistEstimate`). Given a probabilistic program P, a set S of initial states, the number of iterations k, a candidate T expressed as a CNF, parameters $\epsilon, \delta \in (0,1]$, `DistEstimate` outputs an ϵ-additive estimate of the $d(T, cl_k(S))$ with probability at least $1-\delta$. Also, `DistEstimate` requires at most $\lceil \frac{1}{2\epsilon^2} \log(\frac{2}{\delta}) \rceil$ samples from $\mathrm{Unif}(S \times \mathcal{P}(R))$.

Algorithm 1 `IsNotWitness`(T, w)

Initialize $\tau \leftarrow 0$.
$\tau \leftarrow T(w)$
Output $\neg\tau$.

Algorithm 2 `DistEstimate`$(P(V, R), S, T, \epsilon, \delta, k)$

1: Initialize $m \leftarrow \lceil \frac{1}{2\epsilon^2} \log(\frac{2}{\delta}) \rceil, S_U \leftarrow \emptyset, \hat{d}_{S_U} \leftarrow 0, \tau \leftarrow 0$.
2: $S_U \leftarrow m$ iid samples from $\mathrm{Unif}(S \times \mathcal{P}(R))$.
3: **for** $i \in [m]$ **do**
4: With $(x_i, R_i) \in S_U$ as initial state, run the program P for k iterations to obtain an output state y_i.
5: $\tau \leftarrow$ `IsNotWitness`(T, y_i)
6: $\hat{d}_{S_U} \leftarrow \hat{d}_{S_U} + \frac{\tau}{m}$
7: **end for**
8: Output $\hat{d}_{S_U}$.

For a given candidate T, we can write the distance of T from the k-step invariant $cl_k(S)$ as follows: $d(T, cl_k(S)) = \Pr_{x \sim_U S}[(cl_k(S) \setminus T)$ is reachable from x in at most k iterations$]$.

Proof of correctness of `DistEstimate`:

Claim: Given a probabilistic program P, a set S of initial states, the number of iterations k, a candidate T expressed in CNF, parameters $\epsilon, \delta \in (0, 1]$, `DistEstimate` outputs an estimate $\hat{d}_{S_U}$ of the distance $d(T, cl_k(S))$ with the following guarantees:

$$\Pr[|\hat{d}_{S_U} - d(T, cl_k(S))| \leq \epsilon] \geq (1 - \delta)$$

Moreover, the objective of `DistEstimate` is to generate a set of positive counterexamples, i.e., those states which have not been learnt by the candidate but they do actually belong to the true k-step invariant $cl_k(S)$.

Proof. **Description of `IsNotWitness`:** `IsNotWitness` takes in a CNF formula T and a given assignment w of the variables $V \in \mathrm{supp}(T)$ and returns 1 if w is not a witness for T, and 0 otherwise.

Description of `DistEstimate`: Line 2 samples m states from $(S \times \mathcal{P}(R))$ uniformly at random so as to obtain a sample set $S_U = \{(x_1, R_1), (x_2, R_2), ..., (x_m, R_m)\}$, where for each $i \in [m]$, x_i is the initial state and R_i is the fixed initial seed for the internal random bits of P.

Starting from each state defined by $(x_i, R_i); i \in [m]$, the program P is executed for exactly k iterations in Line 4. According to assumption , if for a given initial state (x_i, R_i), the k-length transition to output state y_i goes out of the set represented by T at some j-th iteration ($j \in [k]$), y_i is guaranteed not to be a witness for the candidate T. Hence, it is sufficient to just check the output state y_i reached after k iterations.

Now, let's define the following event for each state $(x_i, R_i) \in S_U; i \in [m]$:

E_i : With (x_i, R_i) as initial state, the output state y_i reached after k iterations is not a witness for T.

Note that the collection of events $\{E_i\}_{i=1}^m$ are mutually independent. Next, we define indicator random variables for these events as $\mathbf{1}_{E_i}$. Define the statistic $\hat{d}_{S_U} = \frac{1}{m} \sum_{i=1}^m \mathbf{1}_{E_i}$. Line 6 updates the estimate $\hat{d}_{S_U}$ based on the check for non-witness of output state y_i for the candidate T in line 5 via the subroutine `IsNotWitness`.

We observe that $\mathbb{E}[\hat{d}_{S_U}] = \Pr_{x \sim_U S}[(cl_k(S) \setminus T)$ is reachable from $x] = d(T, cl_k(S))$ and thus, $\hat{d}_{S_U}$ is an unbiased estimator of the quantity $d(T, cl_k(S))$. Applying additive Chernoff bound given an error parameter $\epsilon \in (0, 1]$, we get

$$\Pr[|\hat{d}_{S_U} - \mathbb{E}[\hat{d}_{S_U}]| \geq \epsilon] = \Pr[|\hat{d}_{S_U} - d(T, cl_k(S))| \geq \epsilon] \leq 2e^{-2m\epsilon^2}$$

We want to make this probability go below a certain threshold, given by δ. Thus,

$$2e^{-2m\epsilon^2} \leq \delta \implies m \geq \frac{1}{2\epsilon^2} \log(\frac{2}{\delta}).$$

This gives us a sample complexity of $O(\frac{1}{\epsilon^2} \log(\frac{2}{\delta}))$. Thus, we can conclude that if we take atleast $\lceil \frac{1}{2\epsilon^2} \log(\frac{2}{\delta}) \rceil$ iid samples from $\text{Unif}(S \times \mathcal{P}(R))$, `DistEstimate` outputs an ϵ-additive estimate $\hat{d}_{S_U}$ of $d(T, cl_k(S))$ with probability at least $1 - \delta$.

□

Algorithm 3 `UnreachSAT`(Υ)

```
Initialize τ ← 0.
τ ← CryptoMiniSat(Υ).
Output ¬τ.
```

Algorithm 4 `Validifier`$(P(V, R), S, T, F, \epsilon, \delta, k)$

```
Initialize l ← ⌈(1/(2ε²)) log(2/δ)⌉, Υ ← {}, i, j ← 0, D_T ← ∅, τ, d_v ← 0.
D_T ← l iid samples from T.
for σ_i ∈ D_T do
    for j ∈ [k] do
        Construct the formula Υ = S ∧ F^j ∧ σ_i.
        τ ← τ ∨ UnreachSAT(Υ).
    end for
    d_v ← (d_v + τ/l).
    τ ← 0.
end for
Output d_v.
```

1.1.1 Analysis of `Validifier`

Motivation: The candidate T generated by the decision tree learner `TreeLearner` might contain some program states which are actually not reachable from S in k iterations of the program P, i.e., those states which do not belong to the true k-step invariant $cl_k(S)$. These program states which are not in the true k-step invariant but have been learnt by `TreeLearner` need to be penalised. Hence, we introduce a new weight function in order to quantify how good or bad our candidate T overapproximates the true k-step invariant $cl_k(S)$. This weight function can be formally defined as follows :-

Definition 1.1.3. Over-approximating weight of a given candidate T: Given a probabilistic program P and a set S of initial states, the k-step invariant or the k-iterative closure of P with respect to S is given by the set $cl_k(S) = \{\sigma \in \{0,1\}^{|V|} \mid \sigma \text{ is reachable from } S \text{ in at most } k \text{ iterations of } P\}$. Given a candidate T, the overapproximating weight function for T can be defined as :

$$w(T) = \Pr_{\sigma\sim T}[\sigma \notin cl_k(S)]$$

Claim: Given a probabilistic program P, a set S of initial states, the number of iterations k, a candidate T expressed in CNF, parameters $\epsilon, \delta \in (0,1]$, `Validifier` outputs an estimate $\hat{w}(T)$ of the weight function $w(T)$ with the following guarantees:

$$\Pr[|\hat{w}(T) - w(T)| \leq \epsilon] \geq (1-\delta)$$

Moreover, the objective of `Validifier` is to generate a set of negative counterexamples, i.e., those states which have been learnt by the candidate but they do not actually belong to the true k-step invariant $cl_k(S)$. The over-approximating weight function precisely penalises these kind of states for a given candidate T.

Proof. **Description of `UnreachSAT`:** `UnreachSAT` takes in a CNF formula Υ and returns 1 if Υ is unsatisfiable and 0 otherwise.

Description of `Validifier`:

Line 2 samples l states from the candidate T almost uniformly at random (using an almost-uniform sampler `CMSGen`) from the witnesses of T to obtain a sample set $D_T = \{y_1, y_2, ..., y_l\}$, where for each $i \in [l]$, y_i is the output state whose reachability needs to be checked for any $j \in [k]$ iterations of the program P starting from S.

Line 5 constructs for every $j \in [k]$, the j-step reachability formula from S for the sampled output state y_i, denoted by $\Upsilon = S \wedge F^j \wedge y_i$, where F^j is the CNF formula which is satisfiable by all the valid j-length runs of the program S.

Thus, the CNF formula Υ is satisfiable if and only if there exists a valid j-length transition of the program P starting from some state in S and ending up in the final state y_i, i.e., if y_i is reachable in exactly j iterations of P from S. Hence, given a final state y_i, if the collection of CNF formulas $\{S \wedge F^j \wedge y_i\}_{j=1}^{k}$ is unsatisfiable, then we can conclude that $y_i \notin cl_k(S)$.

Now, let's define the following event for each state $y_i \in D_T; i \in [l]$ and $j \in [k]$:

$$E_i : \text{The collection of reachability formulas } \{S \wedge F^j \wedge y_i\}_{j=1}^{k} \text{ is unsatisfiable.}$$

Thus, the event E_i holds if the sampled final state y_i is reachable in some $j \in [k]$ iterations of P, starting from S. Note that the collection of events $\{E_i\}_{i=1}^{l}$ are mutually independent. Next, we define indicator random variables for these events as $\mathbf{1}_{E_i}$. Define the statistic $\hat{w}(T) = \frac{1}{l}\sum_{i=1}^{l} \mathbf{1}_{E_i}$. Line 8 updates the estimate $\hat{w}(T)$ based on the check for non-witness (Line 6) of the collection of formulas $\{S \wedge F^j \wedge y_i\}_{j=1}^{k}$ generated by the inner loop (Line 5).

We observe that $\mathbb{E}[\hat{w}(T)] = \Pr_{y\sim_U T}[y \text{ is reachable from } S] = w(T)$ and thus, $\hat{w}(T)$ is an unbiased estimator of the quantity $w(T)$. Applying additive Chernoff bound given an error parameter $\epsilon \in (0,1]$, we get

$$\Pr[|\hat{w}(T) - \mathbb{E}[\hat{w}(T)]| \geq \epsilon] = \Pr[|\hat{w}(T) - w(T)| \geq \epsilon] \leq 2e^{-2l\epsilon^2}$$

We want to make this probability go below a certain threshold, given by δ. Thus,

$$2e^{-2l\epsilon^2} \leq \delta \implies l \geq \frac{1}{2\epsilon^2}\log(\frac{2}{\delta}).$$

This gives us a sample complexity of $O(\frac{1}{\epsilon^2}\log(\frac{2}{\delta}))$. Thus, we can conclude that if we take atleast $\lceil\frac{1}{2\epsilon^2}\log(\frac{2}{\delta})\rceil$ iid samples from the CNF formula T, `Validifier` outputs an ϵ-additive estimate $\hat{w}(T)$ of the weight function $w(T)$ with probability at least $1-\delta$. Also, the number of SAT calls required by `Validifier` in order to verify reachability of the sampled final states $= \lceil\frac{k}{2\epsilon^2}\log(\frac{2}{\delta})\rceil$.

$\square$

Problem definition:

<u>Problem</u> **1.1.5. (Approximate learning of the k-step invariant $cl_k(S)$):** Given a program probabilistic P defined on program variables V, a set S of initial states and parameters $k \in \mathbb{N}$ for the number of program iterations, $\epsilon, \delta \in (0, 1]$, output a candidate $\hat{S}_k$ for the k-step invariant $cl_k(S)$ such that $d(\hat{S}_k, cl_k(S)) \leq \epsilon$ with probability at least $1 - \delta$.

High-level overview of the algorithm:

Ideally, the objective is to learn the k-step invariant $cl_k(S)$. However, it is extremely hard. In this context, can we atleast approximate $cl_k(S)$? That is, we want to output some $\hat{S}_k$ such that $d(\hat{S}_k, cl_k(S))$ is as small as possible. An informal sketch of the algorithm to approximately learn $cl_k(S)$ via $\hat{S}_k$ is given below. The algorithm runs in k phases and tries to learn $cl_k(S)$ in a BFS manner, i.e., it starts off with S and learns $cl_1(S), cl_2(S), ..., cl_k(S)$ via the sequence $\hat{S}_1, \hat{S}_2, ..., \hat{S}_k$.

1. **Phase 1:-** Objective is to learn $cl_1(S)$ starting from S.

   - Sample *enough* states from S and run the program P for one iteration to obtain a set of output states.
   - Build a labeled dataset D using these output states such that the states which are in S are labeled 0 and 1 otherwise.
   - Learn a binary decision tree with *bounded size* (according to the size of the formula we want) on D to output a formula φ.
   - If $d(S \vee \varphi, cl_1(S)) \geq \epsilon$, perform *weighted secondary sampling* (counterexample-guided sampling). Extend the dataset D by labeling these newly sampled instances.
   - Output $\hat{S}_1 = S \vee \varphi$.
   - Theoretical guarantees on how close $\hat{S}_1$ is to the actual one-step invariant $cl_1(S)$ depends on the number of samples taken and the restriction on the size of φ, which is actually dictated by the size of the decision tree learnt.

2. **Phase i ; $i \in \{2, 3, ..., k\}$:-** Objective is to learn $\bar{S}_i$ starting from $\hat{S}_{i-1}$.

   - Sample *enough* states from S and run the program P for i iterations to obtain a set of output states.
   - Build a labeled dataset D using these output states such that the states which are in $\hat{S}_{i-1}$ are labeled 0 and 1 otherwise.
   - Learn a binary decision tree with *bounded size* (according to the size of the formula we want) on D. This decision tree would then correspond to the formula φ such that $d(\hat{S}_{i-1} \vee \varphi, \bar{S}_i)$ is minimized.
   - Output $\hat{S}_i = \hat{S}_{i-1} \vee \varphi$.
   - Once again, theoretical guarantees on how close $\hat{S}_i$ is to the actual i-step invariant $\bar{S}_i$ depends on the number of samples taken and the restriction on the size of φ, which is actually dictated by the size of the decision tree learnt.

Algorithm 5 `ApproxInv`$(P(V,R), S, \epsilon, \eta, \delta, k)$

Initialize $t \leftarrow \lceil \textit{something} \rceil, D_t \leftarrow \emptyset, D \leftarrow \emptyset, \hat{d} \leftarrow \infty, T \leftarrow \{\}, W \leftarrow S$.
for $j \in [k]$ **do**
  $D_t \leftarrow t$ iid samples from $\mathrm{Unif}(S \times \mathcal{P}(R))$.
  $D \leftarrow$ `BuildDataset`$(P(V,R), D_t, W, j)$.
  $T \leftarrow$ `TreeLearner`(D).
  $T \leftarrow$ `Validifier`$(P(V,R), W, T)$.
  $\hat{d} \leftarrow$ `DistEstimate`$(P(V,R), S, T, \frac{\epsilon}{k}, \delta, j)$
  while $\hat{d} \leq \frac{\eta}{k}$ **do**
    $D \leftarrow D \cup$ `SecondarySampler`$(P(V,R), S, T)$
    $T \leftarrow$ `TreeLearner`(D).
    $T \leftarrow$ `Validifier`$(P(V,R), W, T)$.
    $\hat{d} \leftarrow$ `DistEstimate`$(P(V,R), S, T, \epsilon, \delta, j)$
  end while
  $W \leftarrow T$.
  $D_t \leftarrow \emptyset$.
  $D \leftarrow \emptyset$.
end for
Output W.

Algorithm 6 `BuildDataset`$(P(V,R), D_t, W, j)$

Initialize $t \leftarrow |D_t|, D \leftarrow \emptyset, \tau \leftarrow 0$.
for $i \in [t]$ **do**
  With $(x_i, R_i) \in D_t$ as initial state, run the program P for j iterations to obtain an output state y_i.
  $\tau \leftarrow$ `IsNotWitness`(W, y_i)
  $D \leftarrow D \cup (y_i, \tau)$
end for
Output D.

Algorithm 7 `TreeLearner`(D)

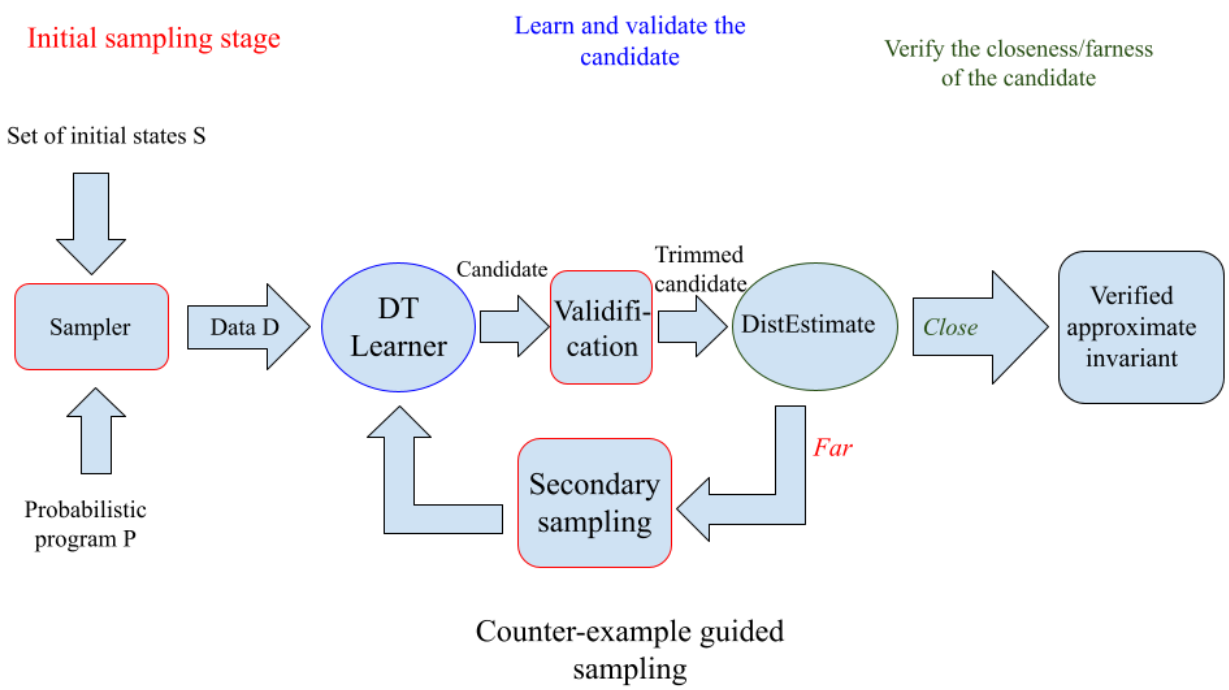
Initialization.
`Split-n-Build`
`Prune`
Output-

Algorithm 8 `SecondarySampler`$(P(V,R), S, T)$

Initialization.

Output-

Figure 1.1: Sketch of `ApproxInv`



1.1.2 Notes on `ApproxInv`:

- `BuildDataset`: This subroutine is used for building a labeled dataset for the decision tree learner `TreeLearner`.
- `IsNotWitness`: This subroutine takes in a CNF formula T and an assignment w of its variables and returns 1 if w is not a witness for T and vice-versa.
- `TreeLearner`: This subroutine is used to learn a candidate invariant based on the labeled dataset returned by `BuildDataset`.
- `Validifier`: This subroutine takes in a candidate formula T and a set W of states and trims T by deleting all those states in T which are not reachable in one step from W.
- `ProgCNF`: It takes in a program P and generates a CNF formula F^P corresponding to the valid runs of the program P.
- `DistEstimate`: Description given in Theorem 1.1.4.
- `SecondarySampler`: This subroutine is meant for counterexample-guided sampling in the CEGIS loop.

#samples	100	500	1000	10000
#times test performed	50	50	50	50
d_{min}	0.11	0.134	0.163	0.1829
d_{max}	0.25	0.20	0.206	0.1962
d_{mean}	0.1731	0.17296	0.18312	0.1896
d_{std}	0.0284	0.01472	0.01042	0.00313

Table 1.1: Performance of `DistEstimate` on a toy program (Ex9: 5 program variables, 4 internal random variables)

1.1.3 Problems in `ApproxInv`:

- `TreeLearner` might learn a small-sized formula T such that there exists some state $s \notin cl_k(S)$ but s is a witness for T. In that case $d(T, cl_k(S)) \to \infty$. *This depends on the pruning scheme of the full decision tree.*

  Arnab: `Validifier` can probably be used to trim these spurious states.

- **We can do the reachability check while sampling.**

1.2 Experimental evaluations on `DistEstimate`: