Path-based SST relies on classical results from labelled combinatorial structures [11] to uniformly sample the set of program paths with length in $[l_{min}, l_{max}]$. Each path sample is provided to a constraint solver (oracle) and labelled as feasible or infeasible; see [9] and references therein. The infeasibility of a given path arises if it violates some dependencies between different parts of the program, referred to as *XOR patterns*. For instance if two *if* nodes are based on an unchanged expression, then their successors are correlated in every feasible path (if the program path includes the *then* successor of the first *if* node, it must also include the *then* successor of the second *if* node).

Because of the small number of available labelled paths (due to the labelling cost) compared to the complexity of the "natural" search space, i.e. that of long strings on a large alphabet, a frugal propositional representation inspired by Parikh maps [12] is considered. For $i = 1 \ldots l_{max}$, let $s[i]$ denote the i-th symbol in string s, set to value $\sigma_{\emptyset}$ if the length of s is less than i.

- To each symbol σ is associated an integer attribute σ_{nb}: $\sigma_{nb}(s)$ is the number of occurrences of symbol σ in path s.
- To the i-th occurrence of a symbol σ is associated a categorical attribute σ_{next^i}. Attribute $\sigma_{next^i}(s)$ gives the next informative[1] symbol following the i-th occurrence of symbol σ in s (or $\sigma_{\emptyset}$ if s contains less than i occurrences of σ).

Preliminary attempts at discriminant learning have been hindered by the tiny percentage of the feasible paths, as could have been expected from [8]. A generative learning approach was then considered.

3 Overview of *EXIST*

This section describes a sampling algorithm called *EXIST* for *Exploration vs eXploitation Inference for Software Testing*, able to retrieve distinct feasible paths with high probability based on a set $\mathcal{S}$ of feasible/infeasible paths, $\mathcal{S}$ initially set to a small set of labelled paths, is gradually enriched with the paths generated by *EXIST* and labelled by the constraint solver.

EXIST proceeds by iteratively exploiting and updating a probabilistic model $\mathcal{P}$. *EXIST* involves two modules: the *learn* module estimates the probability for a path to be feasible conditionally to its extended Parikh description[2]; the *Decision* module uses the $\mathcal{P}$ model to iteratively construct the current path s.

3.1 *Decision* module

Let s (resp. σ) denote the path under construction (resp. the last node symbol in s). Let i be the total number of occurrences of σ in s. Let m be one possible successor node of σ; if m is selected, the total number of m symbols in the final path will be at least the current number of occurrences of m in s, plus one; let j_m denote this number.

Let us define $p_s(m)$ as the probability for a path S to be feasible conditionally to $\mathcal{F}_{s\sigma m}(S) \equiv [\sigma_{next^i}(S) \equiv m] \wedge [\sigma_{nb}(S) \geq j_m]$, estimated by the *learn* module; $p_s(m)$ is conventionally set to 1 if there is no path in $\mathcal{S}$ satisfying $\mathcal{F}_{s\sigma m}$.

Probabilities $p_s(m)$ for m ranging over the successors of σ are used to select the next node in s. Three options have been considered in order to favor the generation of a new feasible path.
The *Greedy* option selects the successor node m maximizing $p_s(m)$.
The *RouletteWheel* option stochastically selects node m with probability proportional to $p(s,m)$.
The *BanditST* option considers the multi-armed bandit problem where every bandit arm corresponds to a successor m of the current node σ and the associated reward is $p_s(m)$, and uses the UCB1 algorithm [1] for determining the best arm/successor node.

3.2 *learn* module

The *learn* module determines how the conditional probabilities used by the *Decision* module are estimated. The baseline *learn* option computes $p_s(m)$ as the fraction of paths in $\mathcal{S}$ satisfying $\mathcal{F}_{s\sigma m}$ that are feasible. However, this option fails to guide *EXIST* efficiently due to the disjunctive nature of the target concept, as shown on the following toy problem.

[1] Formally $\sigma_{next^i}(s)$ is set to $s[pos(i)+1]$, where $pos(i)$ is the index of the i-th occurrence of symbol σ in s; i is initially set to 0, in case σ_{next^i} takes on a constant value over all examples, i is incremented.

[2] This probabilistic model space is meant to avoid the limitations of probabilistic FSAs and Variable Order Markov Models [4]. On one hand, probabilistic FSAs (and likewise simple Markov models) cannot model the long range dependencies of the *XOR patterns*. On the other hand, although Variable Order Markov Models can accommodate such dependencies, they are ill-suited to the sparsity of the initial data available.