MPRI - Final exam - Probabilistic Programming Language
November 2024

Instructions You have 3h to complete the exam. There are 2 independent exercises: one on modeling and one on semantics. The exam is long and you are not expected to answer every question. Questions are numbered 1 to 39. In the margin you find scales of the different part.

- Electronic devices must be turned off and put in your bag.
- Authorized documentation: course slides.
- Your answers can be in French or in English as you wish.
- The questions are numbered: report these numbers on your copy.
- The scale is given on an indicative basis and can be modified.

I). Programming Bayesian models

[4 pt] The purpose of this exercise is to describe a generative model in the area of text analysis based on frequencies.
A *Caesar* cipher encodes a text by shifting characters by a fixed constant given by a key.
For instance with the key C, each letter is shifted by 3 characters and the word "bayes" becomes "edbhv".
Given the frequency table of the letters in English characters and an encrypted text the goal is to learn the key by using inference and probabilistic programming in μ-PPL or byoPPL.

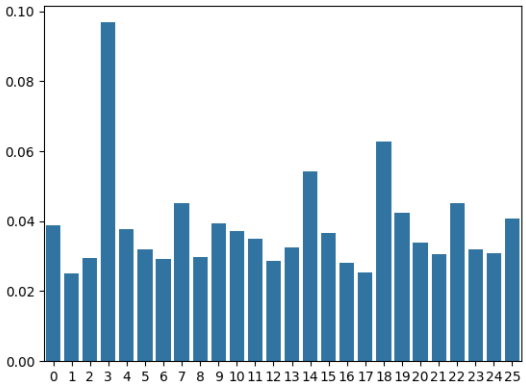
$A : 0.0651738$	$B : 0.0124248$	$C : 0.0217339$	$D : 0.0349835$	$E : 0.1041442$	$F : 0.0197881$
$G : 0.015861$	$H : 0.0492888$	$I : 0.0558094$	$J : 0.0009033$	$K : 0.0050529$	$L : 0.033149$
$M : 0.0202124$	$N : 0.0564513$	$O : 0.0596302$	$P : 0.0137645$	$Q : 0.0008606$	$R : 0.0497563$
$S : 0.051576$	$T : 0.0729357$	$U : 0.0225134$	$V : 0.0082903$	$W : 0.0171272$	$X : 0.0013692$
$Y : 0.0145984$	$Z : 0.0007836$				

Question 1.
Write a probabilistic model `find_key` in the language of your choice (μ-PPL or byoPPL) that takes an encoded text as parameter. Your model should sample uniformly a key, then randomly generate a text according to the frequency table, and finally reject generated texts that do not correspond to the observed data passed as parameter.

Question 2. With an input encoded text of 5 letters, the rejection sampling algorithm with 100 samples does not converge fast enough to produce a result. Explain the algorithm and discuss why it is not satisfactory.

Question 3. Explain the enumeration algorithm and draw the enumeration tree for an input text of 3 letters. Compute the size of the enumeration tree according to the length of the input text. For each sampled key, how many branches satisfy the rejection condition ?

Question 4. With the input text "anicelittletextthatiwouldliketoencode" encoded with a shift of 3 letters, the enumeration sampling algorithm give the following result:



Explain how the implementation can exploit the number of branches satisfying the rejection condition to be able to produce such a result.

Question 5. Assume that the encoded text has been generated with errors and that with probability 0.001, the character is not shifted. Code the corresponding probabilistic model `find_key_error`. Explain why the enumeration algorithm does not converge fast enough anymore.

II). A discrete probabilistic programming language

We introduce a first order probabilistic language with discrete probability and boolean type in Figure 1. Compared to μ-PPL or BYO-PPL, the language only focuses on the probabilistic part: an expression describes a probabilistic model, the `infer` operator is not part of the core language.

A. Semantics

[2 pt] ### A.1. Denotational semantics

The denotational semantics of types and terms is given in Figure 2.

Question 6. Draw the enumeration trees and compute the semantics of the following models (the details of computation are not required).

```
let const = let x = sample(Bernoulli(0.01, name="f") in assume(x)

let cond =
  if sample(Bernoulli(0.5), name="f1")
  then sample(Bernoulli(0.5), name="f2")
  else let () = assume(False) in False

let funny_bernoulli =
  let x = sample(Bernoulli(0.6), name="f1") in
  let y = sample(Bernoulli(0.5), name="f2") in
  let z = x or y
  let () = assume(z) in
  x
```

Question 7. Let $\mu : [\![\tau]\!] \to \mathbb{R}^+$ be a finitely supported discrete measure. Give the formula defining $infer(\mu)$ as the discrete probability distribution given by renormalizing μ.

Question 8. Compute $infer(\,[\![\texttt{const}]\!]\,)$, $infer(\,[\![\texttt{cond}]\!]\,)$ and $infer(\,[\![\texttt{funny\_berrnoulli}]\!]\,)$.

Types

Types τ ::= `Bool` | `unit`

Context G : Variables $\to$ Types for instance: $G = [x \leftarrow \texttt{Bool}, y \leftarrow \texttt{unit}]$

Terms

Variables	x ::= $x, y, z, \ldots$
Values	v ::= `()` | `True` | `False`
Arguments	p ::= () | x
Atomic expressions	a ::= x | v
Expressions	e ::= a | `not` e | e `and` e | e `or` e
	| `if` e `then` e `else` e | `let` p `=` e `in` e | `assume(`e`)`
	| `sample(Bernoulli(`r`), name="`i`")` $\forall r \in \mathbb{R}$
Declarations	d ::= e | `let` f`(`p`)` `=` e `in` d

$$\boxed{\Gamma \vdash e : t} \qquad \frac{}{\Gamma \vdash \texttt{True} : \texttt{Bool}} \qquad \frac{}{\Gamma \vdash \texttt{False} : \texttt{Bool}} \qquad \frac{\Gamma(x) = \tau}{\Gamma \vdash x : \tau}$$

$$\frac{\Gamma \vdash e : \texttt{Bool}}{\Gamma \vdash \texttt{not}\ e : \texttt{Bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \texttt{Bool}}{\Gamma \vdash e_1\ \texttt{and}\ e_2 : \texttt{Bool}} \qquad \frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \texttt{Bool}}{\Gamma \vdash e_1\ \texttt{or}\ e_2 : \texttt{Bool}}$$

$$\frac{\Gamma \vdash c : \tau\ \texttt{dist} \qquad i \in \mathbb{I} \qquad r \in \mathbb{R}}{\Gamma \vdash \texttt{sample(Bernoulli}(r)\texttt{, name="}i\texttt{")} : \texttt{Bool}} \qquad \frac{\Gamma \vdash e : \texttt{Bool}}{\Gamma \vdash \texttt{assume}(e) : \texttt{unit}}$$

$$\frac{\Gamma \vdash e : \texttt{Bool} \qquad \Gamma \vdash e_1 : t \qquad \Gamma \vdash e_2 : t}{\Gamma \vdash \texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : t} \qquad \frac{\Gamma \vdash e_1 : \tau \qquad \Gamma + [x \leftarrow \tau] \vdash e_2 : t_2}{\Gamma \vdash \texttt{let}\ x : \tau = e_1\ \texttt{in}\ e_2 : t_2}$$

FIGURE 1 – Syntax and typing of the discrete first order PPL

$$[\![\texttt{unit}]\!] = \{()\} \qquad [\![\texttt{Bool}]\!] = \{\bot, \top\} \qquad [\![x_1 : \tau_1, \ldots, x_n : \tau_n]\!] = \prod_{i=1}^{n} [\![\tau_i]\!]$$

The Kronecker symbol is defined as $\delta_v(v') = 0$ if $v \neq v'$ and $\delta_v(v) = 1$ otherwise.

$$\boxed{[\![\Gamma \vdash e : \tau]\!] : [\![\Gamma]\!] \to ([\![\tau]\!] \to \mathbb{R}^+)} \qquad [\![\texttt{True}]\!]_\gamma = \delta_\top \qquad [\![\texttt{False}]\!]_\gamma = \delta_\bot \qquad [\![x]\!]_\gamma = \delta_{\gamma(x)}$$

$$[\![\texttt{not}\ e]\!]_\gamma = 1 - [\![e]\!]_\gamma \qquad [\![e_1\ \texttt{and}\ e_2]\!]_\gamma = \mathit{let}\ r = [\![e_1]\!]_\gamma(\top) \times [\![e_2]\!]_\gamma(\top)\ \mathit{in}\ r\delta_\top + (1-r)\delta_\bot$$

$$[\![e_1\ \texttt{or}\ e_2]\!]_\gamma = \mathit{let}\ r = [\![e_1]\!]_\gamma(\top) + [\![e_1]\!]_\gamma(\bot) \times [\![e_2]\!]_\gamma(\top)\ \mathit{in}\ r\delta_\top + (1-r)\delta_\bot$$

$$[\![\texttt{sample(Bernoulli}(r), \texttt{name="}i\texttt{")}]\!]_\gamma = r\delta_\top + (1-r)\delta_\bot \qquad [\![\texttt{assume(}e\texttt{)}]\!]_\gamma = [\![e]\!]_\gamma(\top)\,\delta_()$$

$$[\![\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!]_\gamma = [\![e]\!]_\gamma(\top)\,[\![e_1]\!]_\gamma + [\![e]\!]_\gamma(\bot)\,[\![e_2]\!]_\gamma$$

$$[\![\texttt{let}\ x : \tau = e_1\ \texttt{in}\ e_2]\!]_\gamma = \sum_{v \in [\![\tau]\!]} [\![e_1]\!]_\gamma(v)\,[\![e_2]\!]_{\gamma + [x \leftarrow v]}$$

FIGURE 2 – Denotational Semantics. The unique name associated to each sample site is only used to ensure that random variables are only defined once.

A.2. Weighted Model Counting semantics

Let $\mathcal{V}$ be the finite set of variables that are associated to boolean random choices in a model. In order to factorize the computation of the semantics, we represent the enumeration tree associated to a program as:

- A propositional function φ that associates to each output value v of the program a boolean formula $\varphi(v)$ that characterizes all the branches that lead to this value.
- An accepting formula α that characterizes all the success branches (the one that pass all the assumed observations).
- a weight function ω that associates to each boolean probabilistic choice (sample site) the probability of its true branch.

Given two weight functions ω_1 and ω_2 with disjoint domains $\mathrm{dom}(\omega_1) \cap \mathrm{dom}(\omega_2) = \emptyset$, we denote $\omega_1 \uplus \omega_2$ the weight function $\omega : \begin{cases} z \mapsto \omega_1(z) & \text{when } z \in \mathrm{dom}(\omega_1) \\ z \mapsto \omega_2(z) & \text{when } z \in \mathrm{dom}(\omega_2) \end{cases}$ with domain $\mathrm{dom}(\omega_1) \cup \mathrm{dom}(\omega_2)$.

We introduce the Weighted Formulas Semantics in Figure 3.

Question 9. Give the Weighted Formulas Semantics of `const`, `cond` and `funny_bernoulli` (the intermediate computations are not required).

Question 10. Draw the enumeration tree of `funny_bernoulli` and give an intuitive description of its Weighted Formulas semantics.

Let $\sigma : \mathcal{V} \to \{\bot, \top\}$ be a boolean valuation and denote $\varphi\{\sigma\}$ the closed formula obtained by replacing every variable f by its value $\sigma(f)$ in φ.
A boolean valuation σ models φ, denoted $\sigma \models \varphi$, if $\varphi\{\sigma\} \equiv \top$.

Question 11. Prove that for all valuation σ and boolean expression $\Gamma \vdash e : \texttt{Bool}$,

$$\text{if } v \neq v', \text{ then if } \sigma \models \{\!|e|\!\}^{\varphi}_{\gamma}(v) \text{ then } \sigma \not\models \{\!|e|\!\}^{\varphi}_{\gamma}(v').$$

Let us define the boolean indicator function $\Delta_v = \lambda v'.(v = v')$.

$$
\begin{array}{rcl}
\{\!|\Gamma \vdash e : \texttt{Bool}|\!\} : [\![\Gamma]\!] & \to & ([\![\tau]\!] \to \texttt{Prop}) \times \texttt{Prop} \times \left(\mathcal{V} \to \mathbb{R}^+\right) \\
\{\!|e|\!\}_\gamma & = & (\{\!|e|\!\}^\varphi_\gamma, \{\!|e|\!\}^\alpha_\gamma, \{\!|e|\!\}^\omega) \\
\{\!|\texttt{True}|\!\}_\gamma & = & (\Delta_\top, \top, \emptyset) \\
\{\!|\texttt{False}|\!\}_\gamma & = & (\Delta_\bot, \top, \emptyset) \\
\{\!|x|\!\}_\gamma & = & (\Delta_{\gamma(x)}, \top, \emptyset) \\
\{\!|\texttt{not } e|\!\}_\gamma & = & (\neg \{\!|e|\!\}^\varphi_\gamma, \{\!|e|\!\}^\alpha_\gamma, \{\!|e|\!\}^\omega_\gamma) \\
\{\!|e_1 \texttt{ and } e_2|\!\}_\gamma & = & (\{\!|e_1|\!\}^\varphi_\gamma(\top) \wedge \{\!|e_2|\!\}^\varphi_\gamma \vee \{\!|e_1|\!\}^\varphi_\gamma(\bot) \wedge \Delta_\bot, \\
& & \quad \{\!|e_1|\!\}^\alpha_\gamma \wedge \left(\{\!|e_1|\!\}^\varphi_\gamma(\top) \wedge \{\!|e_2|\!\}^\alpha_\gamma \vee \{\!|e_1|\!\}^\varphi_\gamma(\bot)\right), \\
& & \quad \{\!|e_1|\!\}^\omega_\gamma \uplus \{\!|e_2|\!\}^\omega_\gamma) \\
\{\!|e_1 \texttt{ or } e_2|\!\}_\gamma & = & (\{\!|e_1|\!\}^\varphi_\gamma(\top) \wedge \Delta_\top \vee \{\!|e_1|\!\}^\varphi_\gamma(\bot) \wedge \{\!|e_2|\!\}^\varphi_\gamma, \\
& & \quad \{\!|e_1|\!\}^\alpha_\gamma \wedge \left(\{\!|e_1|\!\}^\varphi_\gamma(\top) \vee \{\!|e_1|\!\}^\varphi_\gamma(\bot) \wedge \{\!|e_2|\!\}^\alpha_\gamma\right), \\
& & \quad \{\!|e_1|\!\}^\omega_\gamma \uplus \{\!|e_2|\!\}^\omega_\gamma) \\
\{\!|\texttt{sample(Bernoulli}(r)\texttt{, name="}i\texttt{")}|\!\}_\gamma & = & (\Delta_{f_i}, \top, [f_i \leftarrow r]) \qquad \text{where } f_i \text{ is fresh} \\
\{\!|\texttt{assume(}e\texttt{)}|\!\}_\gamma & = & (\Delta_{()}, \{\!|e|\!\}^\varphi_\gamma(\top) \wedge \{\!|e|\!\}^\alpha_\gamma, \{\!|e|\!\}^\omega_\gamma) \\
\{\!|\texttt{if } e \texttt{ then } e_1 \texttt{ else } e_2|\!\}_\gamma & = & (\{\!|e|\!\}^\varphi_\gamma(\top) \wedge \{\!|e_1|\!\}^\varphi_\gamma \vee \{\!|e|\!\}^\varphi_\gamma(\bot) \wedge \{\!|e_2|\!\}^\varphi_\gamma, \\
& & \quad \{\!|e|\!\}^\varphi_\gamma(\top) \wedge \{\!|e|\!\}^\alpha_\gamma \wedge \{\!|e_1|\!\}^\alpha_\gamma \vee \{\!|e|\!\}^\varphi_\gamma(\bot) \wedge \{\!|e|\!\}^\alpha_\gamma \wedge \{\!|e_2|\!\}^\alpha_\gamma, \\
& & \quad \{\!|e|\!\}^\omega \uplus \{\!|e_1|\!\}^\omega \uplus \{\!|e_2|\!\}^\omega) \\
\{\!|\texttt{let } x = e_1 \texttt{ in } e_2|\!\}_\gamma & = & (\bigvee_{v \in [\![\tau]\!]} \{\!|e_1|\!\}^\varphi_\gamma(v) \wedge \{\!|e_2|\!\}^\varphi_{\gamma+[x\leftarrow v]}, \\
& & \quad \bigvee_{v \in [\![\tau]\!]} \{\!|e_1|\!\}^\varphi_\gamma(v) \wedge \{\!|e_1|\!\}^\alpha_\gamma \wedge \{\!|e_2|\!\}^\alpha_{\gamma+[x\leftarrow v]}, \\
& & \quad \{\!|e_1|\!\}^\omega \uplus \{\!|e_2|\!\}^\omega)
\end{array}
$$

FIGURE 3 – Weighted Formulas semantics

Let $(\varphi, \alpha, \omega) \in (A \to \texttt{Prop}) \times \texttt{Prop} \times \mathcal{V} \to \mathbb{R}^+$. We define the Weighted Model Counting by

$$\texttt{WMC}(\varphi, \alpha, w)(v) = \sum_{\substack{\sigma \text{ st} \\ \sigma \models \varphi(v) \wedge \alpha}} \prod_{\substack{f \text{ st} \\ \sigma(f) = \top}} w(f) \prod_{\substack{f \text{ st} \\ \sigma(f) = \bot}} (1 - w(f))$$

Question 12. Prove that for every $\omega : \mathcal{V} \to [0, 1]$, $\texttt{WMC}(\Delta_\top, \top, \omega) = 1$.

Question 13. Prove that for any boolean expression with free variables $x_1, \ldots, x_n$, for any boolean values $v_1, \ldots, v_n$, v, $\gamma = [x_1 \leftarrow v_1, \ldots, x_n \leftarrow v_n]$

$$[\![e]\!]_\gamma(v) = \texttt{WMC}(\{\!|e|\!\}^\varphi_\gamma(v), \{\!|e|\!\}^\alpha_\gamma, \{\!|e|\!\}^\omega)$$

The proof is by induction on the structure of e and then by case. Detail the following cases: `True`, x, `sample(Bernoulli(`r`), name="`i`")`, `if` e `then` e_1 `else` e_2, `let` $()$ `=assume(`e_1`) in` e_2

Question 14. Prove that

$$\mathit{infer}([\![e]\!]_\gamma)(v) = \frac{\texttt{WMC}(\{\!|e|\!\}^\varphi_\gamma(v), \{\!|e|\!\}^\alpha_\gamma, \{\!|e|\!\}^\omega)}{\texttt{WMC}(\top, \{\!|e|\!\}^\alpha_\gamma, \{\!|e|\!\}^\omega)}$$

B. Implementation

[8 pt] ### B.1. Enumeration

Our language is restricted to Bernoulli distributions. The support of the posterior distribution is always finite. It is thus possible to compute the posterior distribution using enumeration.

Question 15. Explain how the enumeration algorithm works for probabilistic models expressed in CPS. In particular explain what is the type of the `prob` accumulator, and the behavior of `sample` (the full code is not required).

Question 16. Can you propose an alternative implementation (without CPS) using the unique names associated to each sample site in our language? Compare the two approaches.

Question 17. What is the complexity of the enumeration algorithm. Is it practical, e.g., on the `find_key` model of Section I)?

In the following, we try to make enumeration more scalable by compiling the model to Boolean Decision Diagrams (BDD).

B.2. Boolean Decision Diagrams (BDD)

A Binary Decision Diagram (BDD) is an efficient data structure used to represent a Boolean function. A BDD is a rooted acyclic graph where nodes represent variables (decisions) and the two leaves are the boolean values *true* ($\top$) and *false* ($\bot$). Each variable has two children (typically named *low* and *high*). Given a valuation $\sigma : \mathit{Var} \to \mathit{Bool}$, one can evaluate a BDD starting from the root and then for each variable v follow the low edge if $\sigma(v) = \mathit{false}$, or the high edge if $\sigma(v) = \mathit{true}$.

We define the *if-then-else* operator $x \to y, z$ as:

$$x \to y, z \;=\; (x \wedge y) \vee (\neg x \wedge z)$$

The condition x is called the *test*. An *If-then-else* Normal Form (INF) is a Boolean expression containing only the $\to$ operator and the constants $\bot$ and $\top$, such that tests are only performed on variables (and variables only appear in test conditions).

Question 18. Show that all boolean operators can be expressed in INF. In particular, give the translations of x, $\neg x$, $x \wedge y$, $x \vee y$, and $x \Leftrightarrow y$ where x and y are variables.

Notice that for any boolean formula t containing a variable x,

$$t = x \to t[x \leftarrow \top], t[x \leftarrow \bot] \qquad \text{(Shannon expansion)}$$

Question 19. Using the Shannon expansion, write a systematic translation procedure which returns the INF of a formula by repeatedly applying the Shannon expansion on all variables. The inputs of the translation function are the boolean formula, and an ordered list of all the variables appearing in the formula.

Question 20. Compute the INF of $P = (x_1 \Leftrightarrow y_1) \wedge (x_2 \Leftrightarrow y_2)$ with the variable order x_1, y_1, x_2, y_2.

Question 21. The INF can be represented as a tree where nodes are variables and leaves are the constants $\bot$ and $\top$. What is this tree for P?

Question 22. There should be a lot of identical subtrees in the result of the previous question. Merge all the nodes that can be merged to obtain a Directed Acyclic Graph (DAG) which represents the same formula.

Question 23. What is the DAG for P with the variable order x_1, x_2, y_1, y_2?

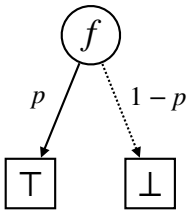


FIGURE 4 – `sample(Bernoulli(p), name="f")`, the plain arrow corresponds to the high edge ($f = \top$), the dotted arrow corresponds to the low edge ($f = \bot$).

More generally, BDD are rooted DAGs where the nodes correspond to the variables with two outgoing edges low and high, and the leaves are the constants $\bot$ and $\top$. In addition a BDD verifies the two following properties:

Ordered: On all paths through the graph the variables respect a given linear order

Reduced: The DAG maximizes sharing (given the variables ordering), i.e.,

1. no two nodes are equivalent (same variable, and same low and high successors),
2. no variable has identical low and high successors.

The DAG of the previous questions should be examples of BDDs.

Question 24. Explain how to build a BDD (ordered AND reduced) given a logical formula and an ordered list of the variables appearing in the formula.

B.3. Weighted BDDs manipulations

A probabilistic model can be represented with a *Weighted* BDD (WBDD) where variables correspond to sample sites and edges are annotated with the transition probability. The WBDD of a sample site is shown in Figure 4. In the following, we write $x \xrightarrow{p} y, z$ for the *weighted if-then-else* operator (with weight p on the high edge and $1 - p$ on the low edge).

WBDDs can be manipulated directly to compute boolean operations. In the following, W is the type of WBDDs, f_i are variables (sample sites), and $t_i : W$ are WBDDs. For the following questions, you can represent WBDDs graphically with a tree (using, e.g., triangle nodes to denote the WBDDs t_i).

Question 25. What is the negation of $f \xrightarrow{p} t_1, t_2$?

Question 26. How to merge $(f \xrightarrow{p} t_1, t_2) \wedge (f \xrightarrow{p} t_3, t_4)$?

Question 27. How to merge $(f_1 \xrightarrow{p_1} t_1, t_2) \wedge (f_2 \xrightarrow{p_2} t_3, t_4)$ in a graph with root f_1?

Question 28. Using the previous questions, write a function `and_bdd` : $(W \times W) \to W$ which computes $t_1 \wedge t_2$.

Question 29. Write a function `or_bdd`: $(W \times W) \to W$ which computes $t_1 \vee t_2$ (Hint: use De Morgan's laws).

B.4. Compilation

We now have everything to compile a model written in our language to two WBDDs φ (model) and α (accepting formula) corresponding respectively to $[\![e]\!]^{\varphi}$ and $[\![e]\!]^{\alpha}$ of Section A.2. In the following $\rho : \mathcal{V} \to W$ is an environment mapping variable names to a WBDD. Given an environment ρ, the compilation returns the WBDDs φ and α:

$$< e, \rho > \rightsquigarrow \varphi, \alpha$$

Question 30. What is the compilation of a sample expression?

$$\frac{?}{< \texttt{sample}(\texttt{Bernoulli}(r), \texttt{name="}i\texttt{"}), \rho > \rightsquigarrow ?, ?}$$

Question 31. What is the compilation of a conditioning expression?

$$\frac{?}{< \texttt{assume}(v), \rho > \rightsquigarrow ?, ?} \qquad \frac{?}{< \texttt{assume}(x), \rho > \rightsquigarrow ?, ?}$$

Question 32. What is the compilation of a local definition?

$$\frac{?}{< \texttt{let}\ x : \tau = e_1\ \texttt{in}\ e_2, \rho > \rightsquigarrow ?, ?}$$

Question 33. Using the result of the previous questions, write the compilation function `compile` which compiles the AST of a model e into the two WBDDs φ and α. We assume that every language construct corresponds to a datatype, e.g., `Var(x)`, `And(e1, e2)`, `IfThenElse(a, e1, e2)`, `Sample(c, name)` etc.

Question 34. Compile the `funny_bernoulli` model of Section A.1 into two WBDDs φ_{fb} and α_{fb}.

B.5. Inference

The last piece is to implement Weighted Model Counting on a WBB to perform inference. Weighted Model Counting can be implemented in a single bottom-up pass of the WBDD. Starting with a weight of 1 for the $\top$ leaf and 0 for the bottom leaf.

Question 35. Write the function `wmc`: $W \to [0, 1]$ which implements Weighted Model Counting for a WBB: `wmc(a)` returns the probability of $\top$.

Question 36. For an expression e without free variables such that $< e, \emptyset > \rightsquigarrow \varphi, \alpha$, how can we compute $\texttt{WMC}([\![e]\!]^{\varphi}_{\gamma}(v), [\![e]\!]^{\alpha}_{\gamma}, [\![e]\!]^{\omega})$ and $\texttt{WMC}(\top, [\![e]\!]^{\alpha}_{\gamma}, [\![e]\!]^{\omega})$ from Section A.2 using `wmc`?

Question 37. Using the previous question and the definitions of Section A.2, write the `infer` function to implement exact inference.

Question 38. Check that `infer(funny_bernoulli)` computes the expected result on the example of Section A.1.

Question 39. What is the complexity of this implementation? Compare it to the result of Section B.1.