# *Discrete Probabilistic Programming Languages*[1]

*Steven Holtzen*
*s.holtzen@northeastern.edu*

*September 29, 2023*

[1] CS7470 Fall 2023: Foundations of Probabilistic Programming.

- Goal for today: compile a simple discrete PPL to BDD

## 1 Compositional compilation of BDDs

- So far we have compiled by inducting on variables
- **Problem**: this is not *compositional*! A compilation process is compositional if it works by compiling big programs out of smaller sub-programs. I.e., it would have a rule that looks something like:

$$\frac{\alpha \rightsquigarrow \alpha' \qquad \beta \rightsquigarrow \beta'}{\alpha \wedge \beta \rightsquigarrow \alpha' \times \beta'}$$

- Compositional compilation is great: gives us modular reasoning about performance, ... (other reasons?)
- **Goal**: Design a compositional process for compiling $\textsc{Prop}_S$ to $\textsc{Bdd}$
- How can we build big BDDs out of smaller ones? Define a way to compose together BDDs, again using a type-directed step-relation $\Gamma \vdash \alpha_1 \wedge \alpha_2 \Downarrow \beta$, shown in Figure 1.

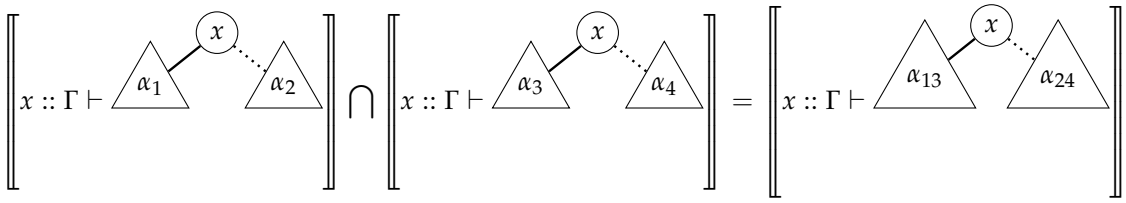
  **Theorem 1** (Correctness)**.** *If* $\Gamma \vdash \alpha_1$, $\Gamma \vdash \alpha_2$, *and* $\Gamma \vdash \alpha_1 \wedge \alpha_2 \Downarrow \beta$, *then* $[\![\alpha_1]\!] \cap [\![\alpha_2]\!] = [\![\beta]\!]$.

  *Proof.* By simultaneous structural induction on syntax of BDDs (note that we need to perform simultaneous induction since there are two BDDs at play here). The rules in Figure 1 are exhaustive (i.e., every pair of syntactic BDDs matches exactly one of these structural rules), so we can proceed by case analysis on each of the compilation rules. The base cases are quite simple and we elide them here. The interesting cases are the inductive cases.

  We will show the case for (SameVarNE). Assume that $\Gamma \vdash \alpha_1 \wedge \alpha_3 \Downarrow \alpha_{13}$ and $\Gamma \vdash \alpha_2 \wedge \alpha_4 \Downarrow \alpha_{24}$. As usual, our inductive hypothesis assumes that the theorem holds for compiled subterms:

  - $[\![\Gamma \vdash \alpha_1]\!] \cap [\![\Gamma \vdash \alpha_3]\!] = [\![\Gamma \vdash \alpha_{13}]\!]$
  - $[\![\Gamma \vdash \alpha_2]\!] \cap [\![\Gamma \vdash \alpha_4]\!] = [\![\Gamma \vdash \alpha_{24}]\!]$

We want to show that:




We reason forward:

$$\left[\!\!\left[ x :: \Gamma \vdash \alpha_{13} \;\; x \;\; \alpha_{24} \right]\!\!\right] = [x \mapsto \texttt{true}] \otimes [\![\Gamma \vdash \alpha_{13}]\!] \bigcup [x \mapsto \texttt{false}] \otimes [\![\Gamma \vdash \alpha_{24}]\!]$$

$$= [x \mapsto \texttt{true}] \otimes \Big( [\![\Gamma \vdash \alpha_1]\!] \cap [\![\Gamma \vdash \alpha_3]\!] \Big) \bigcup [x \mapsto \texttt{false}] \otimes \Big( [\![\Gamma \vdash \alpha_2]\!] \cap [\![\Gamma \vdash \alpha_4]\!] \Big) \qquad \text{By I.H.}$$

$$= \Big( [x \mapsto \texttt{true}]\, [\![\Gamma \vdash \alpha_1]\!] \cap [x \mapsto \texttt{true}]\, [\![\Gamma \vdash \alpha_3]\!] \Big) \bigcup \Big( [x \mapsto \texttt{false}]\, [\![\Gamma \vdash \alpha_2]\!] \cap [x \mapsto \texttt{false}]\, [\![\Gamma \vdash \alpha_4]\!] \Big) \qquad (\star)$$

$$= \Big( [x \mapsto \texttt{true}]\, [\![\Gamma \vdash \alpha_1]\!] \cup [x \mapsto \texttt{false}]\, [\![\Gamma \vdash \alpha_2]\!] \Big) \bigcap \Big( [x \mapsto \texttt{true}]\, [\![\Gamma \vdash \alpha_1]\!] \cup [x \mapsto \texttt{false}]\, [\![\Gamma \vdash \alpha_4]\!] \Big) \qquad (\dagger)$$

$$= \left[\!\!\left[ x :: \Gamma \vdash \alpha_1 \;\; x \;\; \alpha_2 \right]\!\!\right] \cap \left[\!\!\left[ x :: \Gamma \vdash \alpha_3 \;\; x \;\; \alpha_4 \right]\!\!\right]$$

where $(\star)$ follows from a simple lemma that $\otimes$ distributes over intersection and $(\dagger)$ follows from distributivity properties of union and intersection, in particular the fact that for any sets $A, B, C, D$ it is the case that $(A \cup B) \cap (C \cup D) = (A \cap C) \cup (B \cap D)$.[2]

[2] This set theory property has a nice "proof by Venn-diagram"; draw the Venn-diagram of these sets to see this fact clearly.

□

- We would also like to ensure that our compilation rules produce reduced and ordered BDDs. We can formalize this with a type-preservation theorem:

  **Theorem 2** (Type preservation). *If* $\Gamma \vdash \alpha_1$, $\Gamma \vdash \alpha_2$, *and* $\Gamma \vdash \alpha_1 \wedge \alpha_2 \Downarrow \alpha$, *then* $\Gamma \vdash \alpha$.

- These rules are called *bottom-up BDD compilation* [Darwiche and Marquis, 2002, Oztok and Darwiche, 2015].
- There are several other compositional BDD operations we won't have time to explain in lecture, but you can use in your lab, like disjunction, negation, substitution, and existential quantification.
- Why might one prefer one mode of compilation over the other? Do they have different runtime cost? *Yes!*

- **Exercise**: Give example families of formulae where top-down compilation is faster than bottom-up and vice-versa.

## 2 DISC: *A simple discrete PPL*

- Syntax:

```
  e ::=
  | x ← e; e
  | observe e; e
  | flip q                          // q is a rational value
  | if e then e else e
  | return e
  | true | false
  | e ∧ e | e ∨ e | ¬ e |
  | ( e )
p ::= e
```

- DISC looks very similar to a standard functional programming language, but has two some interesting new keywords: `flip`, `observe`, and `return`
- `flip` $\theta$ allocates a new random quantity that is `true` with probability $\theta$ and `false` with probability $1 - \theta$
- `return e` turns a non-probabilistic quantity into a probabilistic one, i.e. `return true` is a *probabilistic quantity* that is `true` with probability 1 and `false` with probability 0
- Example program and its interpretation:

```
x ← flip 0.5;
y ← flip 0.5;
return x ∧ y
```

  This program outputs the probability distribution $[\texttt{true} \mapsto 0.25, \texttt{false} \mapsto 0.75]$.

- `observe` is a powerful keyword that lets us *condition* the program. For instance, suppose I want to model the following scenario: "flip two coins and observe that at least one of them is heads. What is the probability that the first coin was heads?"

  We can encode this scenario as a DISC program:

```
x ← flip 0.5;
y ← flip 0.5;
observe x ∨ y;
return x
```

  This program outputs the probability distribution:

$$[\texttt{true} \mapsto (0.25 + 0.25)/0.75, \texttt{false} \mapsto 0.25/0.75]$$

- **Type system**: terms can either be pure Booleans of type $\mathbb{B}$ or distributions on Booleans of type $\texttt{Dist}(\mathbb{B})$. So, we have the following type definition:

$$\tau ::= \mathbb{B} \mid \texttt{Dist}(\mathbb{B}). \tag{1}$$

- We define a typing judgment $\Gamma \vdash \texttt{e} : \tau$ that associates each term with a type. The typing context $\Gamma$ is a map from identifiers to types.

$$\Gamma \vdash \texttt{true} : \mathbb{B} \qquad \Gamma \vdash \texttt{false} : \mathbb{B} \qquad \Gamma \vdash \texttt{flip}\ \theta : \texttt{Dist}(\mathbb{B})$$

$$\frac{\Gamma \vdash \texttt{e} : \mathbb{B}}{\Gamma \vdash \texttt{return e} : \texttt{Dist}(\mathbb{B})} \qquad \frac{\Gamma \vdash \texttt{e}_1 : \mathbb{B} \qquad \Gamma \vdash \texttt{e}_2 : \tau}{\Gamma \vdash \texttt{observe e}_1\texttt{; e}_2 : \tau}$$

$$\frac{\Gamma \vdash \texttt{e}_1 : \texttt{Dist}(\mathbb{B}) \qquad \Gamma \cup [x \mapsto \mathbb{B}] \vdash \texttt{e}_2 : \tau}{\Gamma \vdash x \leftarrow \texttt{e}_1\texttt{; e}_2 : \tau}$$

$$\frac{\Gamma \vdash \texttt{e}_1 : \mathbb{B} \qquad \Gamma \vdash \texttt{e}_2 : \tau \qquad \Gamma \vdash \texttt{e}_3 : \tau}{\Gamma \vdash \texttt{if e}_1\ \texttt{then e}_2\ \texttt{else e}_3 : \tau} \qquad \frac{\Gamma \vdash \texttt{e}_1 : \mathbb{B} \qquad \Gamma \vdash \texttt{e}_2 : \mathbb{B}}{\Gamma \vdash \texttt{e}_1 \wedge \texttt{e}_2 : \mathbb{B}}$$

*Denotational semantics of* Disc

- Associates each term with an *unnormalized probability distribution* (i.e., the total probability mass may be less than 1).
- Has the type $[\![\texttt{e}]\!] : \texttt{Bool} \to [0, 1]$, and has the following inductive definition:

$$[\![\texttt{flip}\ \theta]\!](v) = \begin{cases} \theta & \text{if } v = T \\ 1 - \theta & \text{if } v = F \end{cases}$$

$$[\![\texttt{return}\ e]\!](v) = \begin{cases} 1 & \text{if } v = [\![e]\!] \\ 0 & \text{otherwise} \end{cases}$$

$$[\![x \leftarrow e_1; e_2]\!](v) = \sum_{v'} [\![e_1]\!](v') \times [\![e_2[x \mapsto v']]\!](v)$$

$$[\![\texttt{observe}\ e_1; e_2]\!](v) = \sum_{\{v' \mid [\![e_1(v')=T\}]\!]} [\![e_2]\!](v)$$

- The semantics for non-probabilistic terms is standard. The semantic evaluation has type $[[e]] : \texttt{Bool}$ for closed terms.

- These semantics give an unnormalized distribution. The main semantic object of interest is the normalized distribution, which is given by the **normalized semantics**:

$$[\![e]\!]_D(T) = \frac{[\![e]\!](T)}{[\![e]\!](T) + [\![e]\!](F)},$$

defined analogously for the false case.

### *Compiling* Disc *to* Prop

- **Goal**: give a semantics-preserving compilation $\rightsquigarrow$ that compiles Disc to Prop
- In order to handle observations, we will compile Disc programs into *two* Prop programs: one that computes the unnormalized probability of returning `true`, and one that computes the probability of evidence (i.e. normalizing constant)
- Inductive description has the shape $\mathtt{e} \rightsquigarrow (\mathtt{p}_1, \mathtt{p}_2)$. We want to define this relation to satisfy the following **adequacy condition**:

$$[\![\mathtt{e}]\!]_D(\mathtt{true}) = \frac{[\![\mathtt{p}_1]\!]}{[\![\mathtt{p}_2]\!]}. \tag{2}$$

We will shorten this description to $\mathtt{e} \rightsquigarrow (\varphi, \varphi_A, w)$ and assume that the two formulae share a common $w$. The adequacy condition then becomes:

$$[\![\mathtt{e}]\!]_D(\mathtt{true}) = \frac{[\![(\varphi, w)]\!]}{[\![(\varphi_A, w)]\!]}. \tag{3}$$

- Compilation relation:

$$\mathtt{true} \rightsquigarrow (\mathtt{true}, \mathtt{true}, \emptyset) \qquad \mathtt{false} \rightsquigarrow (\mathtt{false}, \mathtt{true}, \emptyset)$$

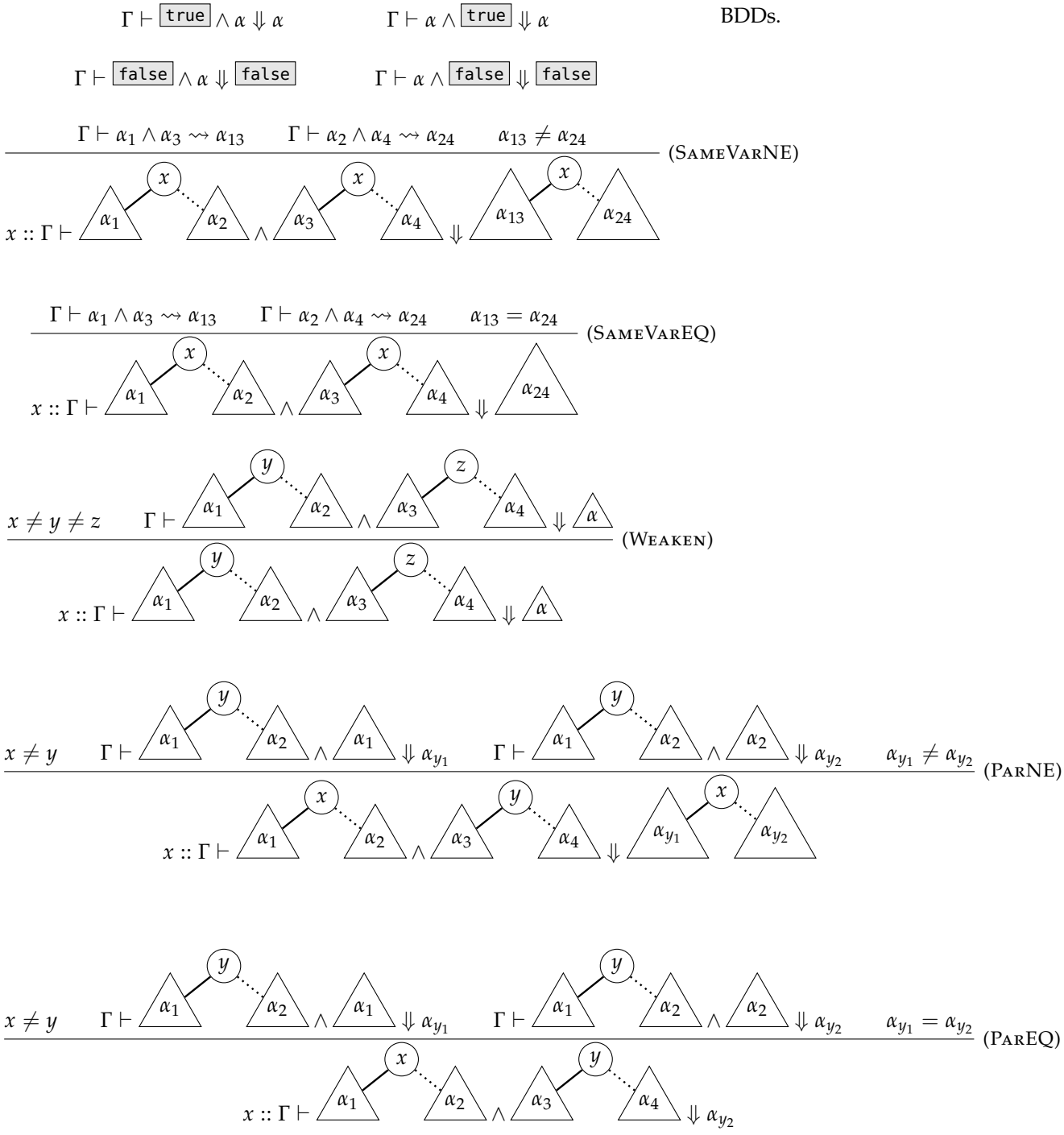
$$x \rightsquigarrow (x, \mathtt{true}, \emptyset) \qquad \frac{\mathtt{fresh}\ x}{\mathtt{flip}\ \theta \rightsquigarrow (x, \mathtt{true}, [x \mapsto \theta, \overline{x} \mapsto 1-\theta])}$$

$$\frac{\mathtt{e}_1 \rightsquigarrow (\varphi, \varphi_A, w) \qquad \mathtt{e}_2 \rightsquigarrow (\varphi', \varphi'_A, w')}{x \leftarrow \mathtt{e}_1; \mathtt{e}_2 \rightsquigarrow (\varphi'[\varphi/x], \varphi'_A[\varphi/x] \wedge \varphi_A, w_1 \cup w_2)}$$

$$\frac{\mathtt{e}_1 \rightsquigarrow (\varphi, \varphi_A, w) \qquad \mathtt{e}_2 \rightsquigarrow (\varphi', \varphi'_A, w')}{\mathtt{observe}\ \mathtt{e}_1; \mathtt{e}_2 \rightsquigarrow (\varphi', \varphi'_A \wedge \varphi_A, w_1 \cup w_2)}$$

**Theorem 3** (Adequacy)**.** *For well-typed term* $\mathtt{e}$*, assume* $\mathtt{e} \rightsquigarrow (\varphi, \varphi_A, w)$*. Then,* $[\![\mathtt{e}]\!]_D(\mathtt{true}) = [\![(\varphi, w)]\!] / [\![(\varphi_A, w)]\!]$.

Figure 1: Rules for compiling BDDs.



## *References*

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

Umut Oztok and Adnan Darwiche. A top-down compiler for sentential decision diagrams. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.