Discrete Probabilistic Programming Languages[1]

Steven Holtzen
s.holtzen@northeastern.edu

October 2, 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:

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

- 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 $\Gamma \vdash \alpha_1 \land \alpha_2 \Downarrow \beta$, shown in Figure 1.

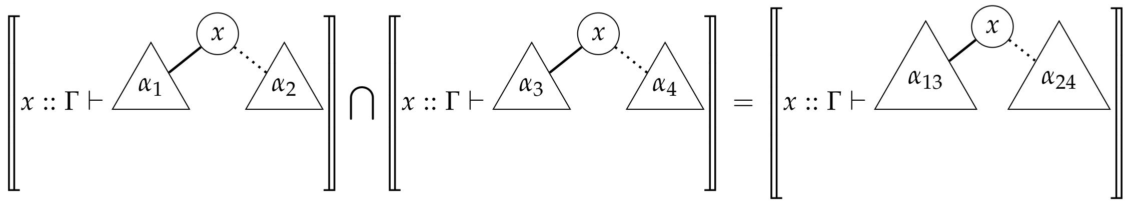
  Theorem 1 (Correctness). *If $\Gamma \vdash \alpha_1$, $\Gamma \vdash \alpha_2$, and $\Gamma \vdash \alpha_1 \land \alpha_2 \Downarrow \beta$, then $[\![\alpha_1]\!] \cap [\![\alpha_2]\!] = [\![\beta]\!]$.*

  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.

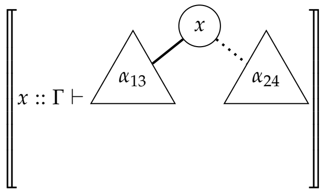
  We will show the case for (SameVarNE). Assume that $\Gamma \vdash \alpha_1 \land \alpha_3 \Downarrow \alpha_{13}$ and $\Gamma \vdash \alpha_2 \land \alpha_4 \Downarrow \alpha_{24}$. As usual, our inductive hypothesis assumes that the theorem holds for compiled subterms:

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

We want to show that:



We reason forward:



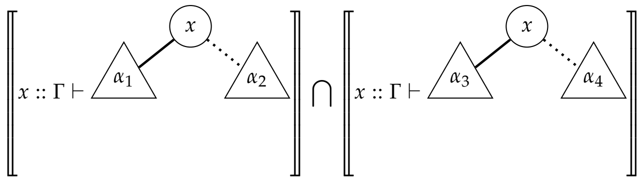
$$= [x \mapsto \texttt{true}] \otimes [\![\Gamma \vdash \alpha_{13}]\!] \bigcup [x \mapsto \texttt{false}] \otimes [\![\Gamma \vdash \alpha_{24}]\!]$$

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

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

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

$=$



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.

$\square$

- 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.

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.

$$\Gamma \vdash \boxed{\texttt{true}} \wedge \alpha \Downarrow \alpha \qquad \Gamma \vdash \alpha \wedge \boxed{\texttt{true}} \Downarrow \alpha$$

$$\Gamma \vdash \boxed{\texttt{false}} \wedge \alpha \Downarrow \boxed{\texttt{false}} \qquad \Gamma \vdash \alpha \wedge \boxed{\texttt{false}} \Downarrow \boxed{\texttt{false}}$$

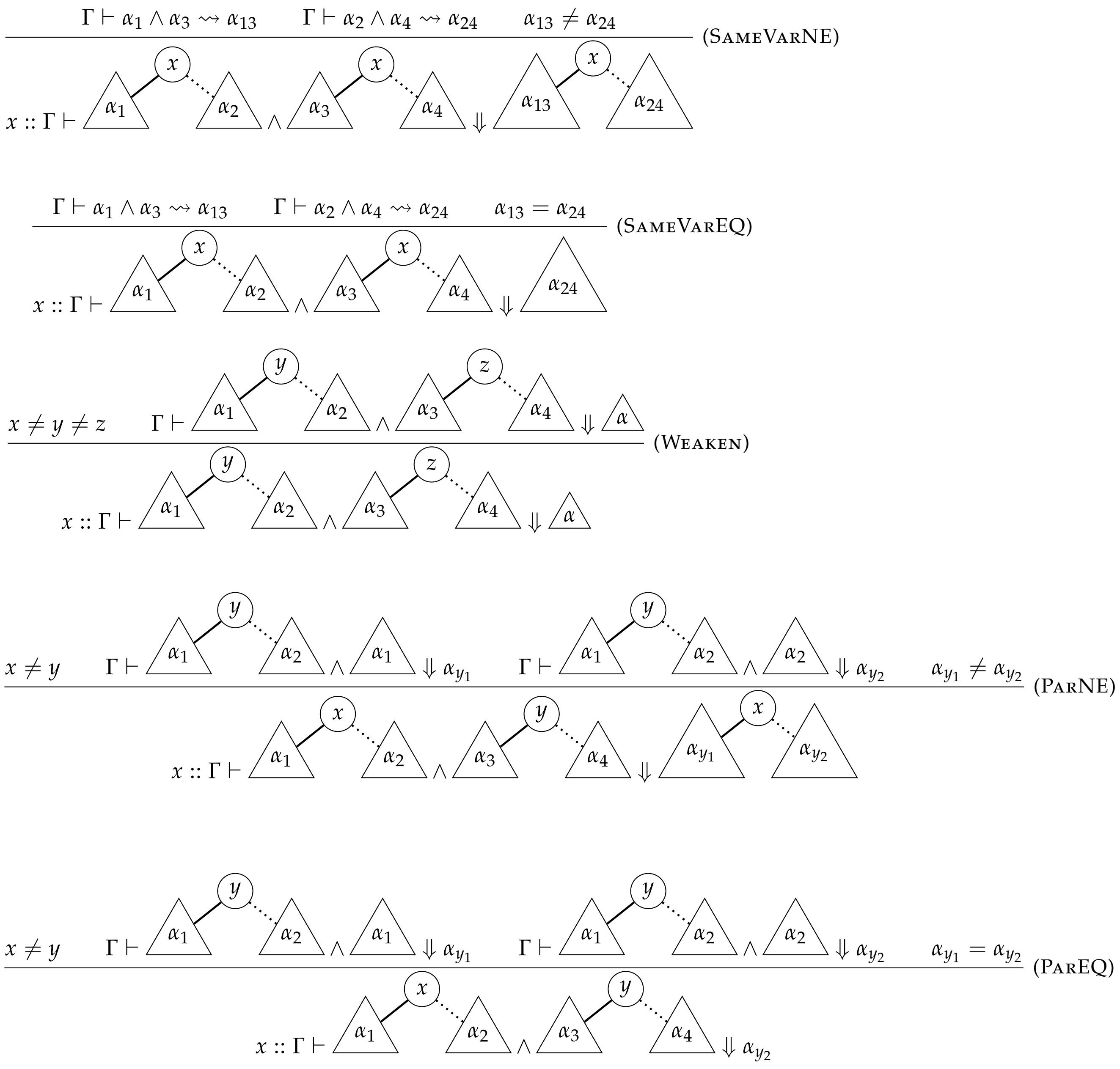

$\Gamma \vdash \alpha_1 \wedge \alpha_3 \rightsquigarrow \alpha_{13}$
$\Gamma \vdash \alpha_2 \wedge \alpha_4 \rightsquigarrow \alpha_{24}$
$\alpha_{13} \neq \alpha_{24}$
(SameVarNE)
$\alpha_{13} = \alpha_{24}$
(SameVarEQ)
$x \neq y \neq z$
(Weaken)
$x \neq y$
$\alpha_{y_1} \neq \alpha_{y_2}$
(ParNE)
$\alpha_{y_1} = \alpha_{y_2}$
(ParEQ)
$x :: \Gamma \vdash$

Figure 1: Rules for compiling BDDs.