Propositional Probability IV: Binary Decision Diagrams[1]

Steven Holtzen
s.holtzen@northeastern.edu

September 29, 2023

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

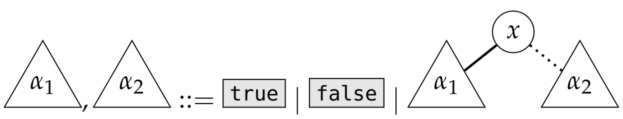
- Note: not class next Tuesday, Sept 26 (Software Day)
- Join Slack, I make announcements there

1 *Binary decision diagrams (BDDs)*

BDDs were introduced by Bryant [1992]. Meinel and Theobald [1998] is a good reference for learning how to implement and analyze BDDs. Knuth [2013] has a comprehensive discussion on BDDs as well.

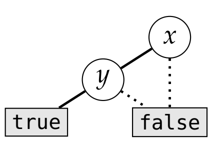
- **Problem**: $\textsc{Prop}_S$ is not very expressive
- **Goal:** More expressive tractable languages
- What can we improve about $\textsc{Prop}_S$? Observe that there may be *repetitious sub-syntactic terms*, and we'd like to exploit those, just like we observed in our memoized reduction scheme
- How can we represent repetitious subterms syntactically? By changing our syntax to permit *directed acyclic graphs*!
- Syntax of **binary decision diagrams** (BDDs):

Here we give a BNF grammar that permits directed acyclic graphs as terms. This is quite non-standard: we are implicitly permitting the structural re-use of terms in this grammar, and assuming an induction principle that "inducts on the structure of DAGs" in this definition of syntax. I don't want to get to into the weeds on this, so we will mostly see through examples how we write terms in this language and perform proofs by structural induction. See Braibant et al. [2014] for some discussion on this.

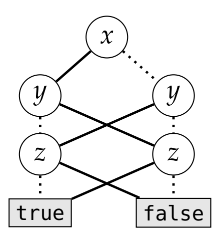


  I.e., each BDD is a rooted directed acyclic graph with terminal nodes labeled `true` and `false`, and internal *branch nodes* labeled by propositional variables. The solid edge of a branch node is called the *high edge*, and the dashed edge is called the *low edge*.

- Here are some example BDDs:



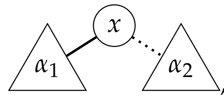
(a) A BDD representing $x \wedge y$



(b) A BDD representing $x \oplus y \oplus z$

Figure 1: Example BDD programs.

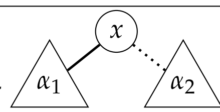
  Solid edges are high edges, dotted edges are low edges. BDDs are read top-down, where the implied directionality of the arrow is top-down.

- Observe that there can many equivalent ways of writing the same BDD for a particular denotation (show examples on board)
- A BDD is called *ordered* if each variable is used *at most once* and in a *fixed order*. We will consider only ordered BDDs.
- A BDD is called *reduced* if, for every node , it is the case that $\alpha_1 \neq \alpha_2$.
- We can enforce these constraints with a typing judgment that is reminescent of *ordered affine linear logic* [Girard, 1987].

  Let Γ be an *ordered* list of propositional variables. A term in ROBDD is a BDD term that satisfies the following typing judgment, written $\Gamma \vdash \triangle$:

$$\Gamma \vdash^r \boxed{\texttt{true}} \qquad \Gamma \vdash^r \boxed{\texttt{false}} \qquad \frac{\Gamma \vdash^r \alpha}{x :: \Gamma \vdash^r \alpha}\ (\textsc{Weaken})$$

$$\Gamma \vdash^r \alpha_1 \qquad \Gamma \vdash^r \alpha_2 \qquad \alpha_1 \neq \alpha_2$$

$x :: \Gamma \vdash^r$ 

Theorem 1 (Canonicity of ROBDD). *For two well-typed ROBDD terms $\Gamma \vdash^r \alpha_1$ and $\Gamma \vdash^r \alpha_2$, it is the case that $[\![\Gamma \vdash \alpha_1]\!] = [\![\Gamma \vdash \alpha_2]\!]$ if and only if $\alpha_1 = \alpha_2$.*

Proof. It is straightforward to show that if $\alpha_1 = \alpha_2$ then $[\![\Gamma \vdash^r \alpha_1]\!] = [\![\Gamma \vdash^r \alpha_2]\!]$. The proof in the other direction follows by inducting on $|\Gamma|$.

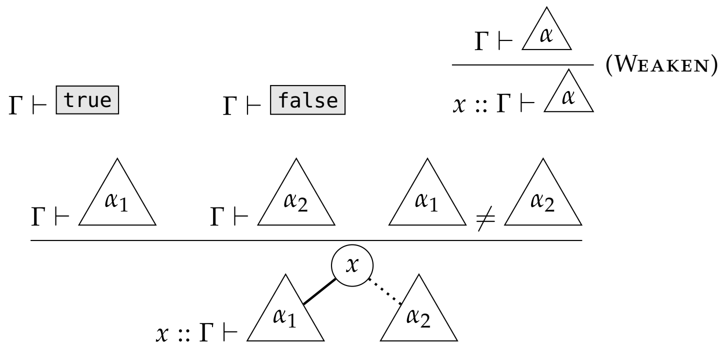
Base case: Assume $|\Gamma| = 0$. Assume $[\![\emptyset \vdash \alpha_1]\!] = [\![\emptyset \vdash \alpha_2]\!]$. By our typing judgment for ROBDD, α_1 and α_2 must each be either `true` or `false`; these two cases have different semantics (either $\emptyset$ or $\{\top\}$), so we can conclude that this implies that $\alpha_1 = \alpha_2$.

Inductive argument: (come back if time, if no time, see Bryant [1992] and Knuth [2013] for an extended discussion)

$\square$

- We can enforce this ordering constraint with a typing judgment that is reminescent of *ordered affine linear logic* [Girard, 1987].

Let Γ be an *ordered* list of propositional variables. A term in ROBDD is a BDD term that satisfies the following typing judgment, written $\Gamma \vdash \triangle$:



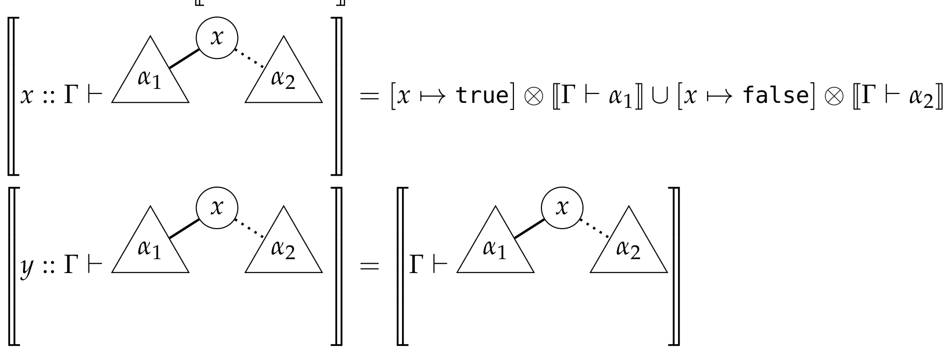
- Semantics of BDDs: Let Γ be an ordered list of propositional variables. For convenience, we define an operation $\otimes$ that adds an assignment to a set of instances, i.e.:

$$[x \mapsto \texttt{true}] \otimes \{[z \mapsto \texttt{false}], [z \mapsto \texttt{true}]\} = \\ \{[x \mapsto \texttt{true}, z \mapsto \texttt{false}], [x \mapsto \texttt{true}, z \mapsto \texttt{true}]\}$$

Formally, we say $[x \mapsto v] \otimes I = \{[x \mapsto v] \cup \gamma \mid \gamma \in I\}$. Then:

$$\left[\!\!\left[\Gamma \vdash \boxed{\texttt{true}}\right]\!\!\right] = [\![\Gamma]\!]$$

$$\left[\!\!\left[\Gamma \vdash \boxed{\texttt{false}}\right]\!\!\right] = \emptyset$$



We define $[\![\Gamma]\!]$ inductively:

$$[\![[]]\!] = \{\top\}$$

$$[\![x :: \Gamma]\!] = [x \mapsto \texttt{true}] \otimes [\![\Gamma]\!] \bigcup [x \mapsto \texttt{false}] \otimes [\![\Gamma]\!]$$

where $\top$ is the empty map.

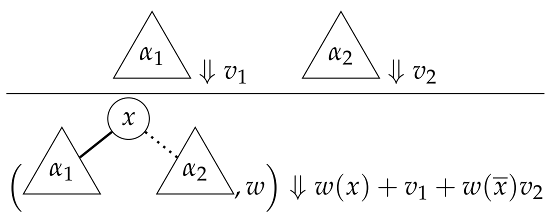
Probabilistic Bdd

- Just like how we made propositional logic, we can define a probabilistic version of BDDs and give them a probabilistic semantics. We call this language Bdd.

- Syntax of Bdd: pairs (α, w), where α is a BDD and w is a map from propositional variables to probabilities

- The denotational semantics are nearly identical to Prop_S, so we skip then here. The operational semantic are quite similar as well:

$$(\boxed{\texttt{true}}, w) \Downarrow 1 \qquad\qquad (\boxed{\texttt{false}}, w) \Downarrow 0$$



- Is the above a a tractable runtime for Bdd? This question is a bit subtle and depends on how we define structural induction on BDDs. For now, let's assume that we are caching as traverse (just like the memoization routine from last time); this memoization routine is a surely a tractable runtime for Bdd.

Theorem 2. Bdd *is more expressive than* Prop_S.

Proof. To show this, we must show two things: (1) that there is an efficient compilation from Prop_S to Bdd, and (2) that there is no efficient compilation from Bdd to Prop_S. Showing (1) is straightforward. Showing (2) is quite tricky.

In general, there are the following strategies we can use to establish that there is no efficient semantics-preserving translation:

- *Denotational lower-bounds*: Show that there is a particular denotation that separates the two languages

- *Reduction*: Show that the existence of a efficient semantics-preserving map can be used to give an efficient algorithm for something we know is hard (i.e., solving SAT)

In this case it is easier to give a denotational lower-bound[2]. We will have the following two lemmas that establish separation between Prop_S and Bdd:

[2] ...but, if you can find a reduction, I'd be interested in seeing it!

Lemma 1. *For any integer N, There exists a BDD of size less than $2n$ whose denotation is equal to* $\left[\!\left[\bigoplus_i^N x_i \right]\!\right]$.

Lemma 2. *For any N there does not exist a $\textsc{Prop}_S$ formula represents the denotation given by $\left[\!\left[\bigoplus_i^N x_i\right]\!\right]$ whose size is less than 2^N.*

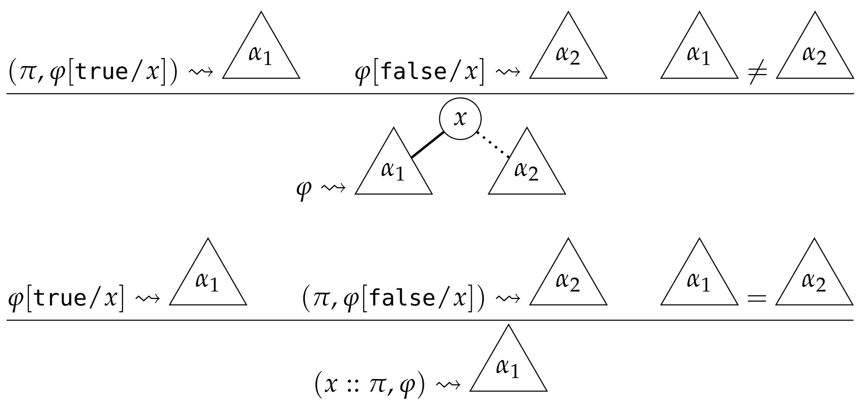
$\square$

Compiling $\textsc{Prop}_S$ *to* Bdd

In the literature this process is typically called *top-down knowledge compilation*. See [Darwiche and Marquis, 2002, Oztok and Darwiche, 2015, Darwiche, 2004].

- **Goal**: Build a compiler from Prop to Bdd. The weight function translation is simple. We focus on translating formulae, which now has has a contextual compilation relation $(\pi, \varphi) \rightsquigarrow$ Bdd, where now ρ is a map from Prop to Bdd:

$$(\pi, \texttt{true}) \rightsquigarrow \boxed{\texttt{true}} \qquad (\pi, \texttt{false}) \rightsquigarrow \boxed{\texttt{false}}$$



-

  Theorem 3. *The above compilation rule produces well-typed BDDs.*

- **Observe**: This looks almost identical to our memoized semantics $\Downarrow^m$. In some sense, the memoized semantics are efficient if and only if there exists a compilation to Bdd. This means that the syntax of Bdd perfectly captures the subset of Prop that can be executed efficiently by $\Downarrow^m$

References

Thomas Braibant, Jacques-Henri Jourdan, and David Monniaux. Implementing and reasoning about hash-consed data structures in coq. *Journal of automated reasoning*, 53(3):271–304, 2014.

Randal E Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys (CSUR)*, 24(3): 293–318, 1992.

Adnan Darwiche. New advances in compiling cnf to decomposable negation normal form. In *Proc. of ECAI*, pages 328–332. Citeseer, 2004.

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

Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1): 1–101, 1987.

Donald E Knuth. *Art of Computer Programming, Volume 4*. Addison-Wesley Professional, 2013.

Christoph Meinel and Thorsten Theobald. *Algorithms and Data Structures in VLSI Design: OBDD-foundations and applications*. Springer Science & Business Media, 1998.

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