COMP60013: Logic-Based Learning Notes

2 Logic and Logical Inference

- Clausal Representation Definitions
  - **Literal**: atomic formula or its negation.
    - It is **ground** if it contains no variables.
    - l' is an **instance** of l, if, for some substitution θ, $l' = l\theta$.
  - **Clause**: disjunction of one or more literals.
    - **Horn clauses**: at most one positive literal.
      - $\{h\} \leftarrow b_1, \ldots, b_n$
      - **Definite clauses**: exactly one positive literal.
      - **Denials**: no positive literal.
    - Horn clauses can be extended by permitting atoms in the body of rules to be prefixed with *not*.
      - $\{h\} \leftarrow b_1, \ldots, b_n, not\ b_{n+1}, \ldots, not\ b_m$
      - **Normal clauses**: exactly one positive literal.
      - **Normal denials**: no positive literal.
      - The *not* operator can only be used on *ground* instances.
  - **Theory**: a conjunction of clauses (denoted as a set of clauses).

- Semantic Definitions (w.r.t. KB)
  - **Herbrand Domain** (HD): the **set of all ground terms** formed using only *constants* and *function* symbols that appear in KB.
  - **Herbrand Base** (HB): the **set of all ground atoms** formed using *predicate* symbols in KB and ground terms in the HD.
  - **Herbrand Interpretation** (HI): (any subset of HB) a set of ground atoms formed using *constant, function, predicate* symbols occuring in KB.
  - **Herbrand Model (HM)**: a HI that satisfies all clauses in KB.
    - It is a **Minimal HM** iff none of its subsets is a HM of KB.
    - If KB is a satisfiable set of Horn clauses then there is a unique Minimal HM called the **least HM (LHM)**.
    - The LHM captures the semantics of KB.
  - **Grounding**: set of all ground instances $c\theta$, for $c \in KB$ and unifier θ replacing variables with terms in the HD.
    - Clausal theory KB is satisfiable iff $ground(KB)$ is satisfiable.
    - Note: if fact $a \in ground(KB)$, then all HMs of KB must contain a.

- **Skolemisation**:
  - $\exists X\ p(X) \mapsto p(c)$, for some *new* constant c.
  - $\forall X \exists Y\ p(X, Y) \mapsto \forall X\ p(X, f(X))$, for some *new* function f.

- **Resolution** Procedure
  - Given two clauses $\phi_1 \vee C_1$ and $\neg\phi_2 \vee C_2$,
    - rename variables so they appear distinct in clauses ϕ_1 and $\neg\phi_2$.
    - for any substitution θ with $\phi_1\theta = \phi_2\theta$, infer $(C_1 \vee C_2)\theta$.
  - It is refutation complete ($KB \vdash G$ iff $KB \cup \neg G \vdash [\ ]$).
  - Prolog uses **SLD resolution**
    - It assumes working with *a set of definite clauses* and a *denial*.
    - At each step, a new denial is resolved from a denial and a def. clause.
    - By convention, the *left-most* denial atom, also called the *subgoal*, is chosen for resolution. (no shortcuts - do it sequentially!)
    - By definition, a derivation fails if the last element in the derivation is not an $[\ ]$ and cannot be resolved any further.
  - An extension to SLD for handling normal clauses is **SLDNF**.
    - At any point, when the left-most term is prefixed with *not*:
      1. Begin a new derivation for the negation of the term.
      2. The result (success/failure) is then the opposite of the result inside.
    - To show failure, one should show *all branches* fail.
    - To start a derivation of the negation, the literal must be *ground*!

- **Abduction**
  - An **abductive task** is given by $\langle KB, Ab, IC \rangle$, where
    - KB (Knowledge Bsse): set of normal clauses
    - Ab (Abducibles): set of ground undefined literals
    - IC (Constraints): set of normal denials
  - Given an abductive task and an observation O, an **abductive solution** of O is a set Δ of ground literals such that

    $(i)\ \Delta \subseteq Ab, \quad (ii)\ KB \cup \Delta \vDash O, \quad (iii)\ KB \cup \Delta \nvDash \bot, \quad (iv)\ KB \cup \Delta \vdash IC$

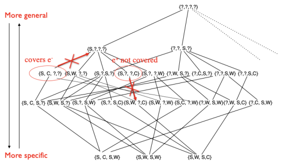
- An abductive proof involves two phases being called alternately:
  1. **Abductive phase**
     a. Set goal to be O and $\Delta = \{\}$.
     b. Use SLDNF to prove the goal by refutation.
     c. At each step, if the subgoal is:
        - **not an abducible**, continue SLDNF.
        - **assumed abducible**, resolve subgoal, continue SLDNF.
        - **abducible not yet assumed** (neither its negation), begin consistency phase with its negation.
     d. Succeed when no further subgoals are left unproven.
  2. **Consistency phase**
     a. Add new assumption to current list Δ_i.
     b. Succeed when failure (black square) is derived.
     c. At each step, if the subgoal is:
        - **not abducible**, continue SLDNF.
        - **assumed abducible**, resolve subgoal, continue SLDNF.
        - **negation of assumed abducible**, fail entire denial.
        - **abducible not yet assumed**, start abductive phase with its negation (but don't add anything to Δ_i).

3 Inductive Logic Programming

- An **inductive logic programming** (ILP) task is a *search problem* involving *minimising a loss function* (the **more general than** relation).

- Given *observations* $\langle E^+, E^- \rangle$, a *background knowledge* B, and a *covers* relation c, a **predictive ILP task** aims to find a hypothesis h with:
  - $c(B, h, e)$, $\forall e \in E^+$ (completeness), $\neg c(B, h, e)$ $\forall e \in E^-$ (consistency)
  - We define $c(B, h, e)$ as $B \cup h \vDash e$ (learning from entailment).
  - We call such an h an **inductive solution**.

- **Concept Learning with a Version Space**
  - Learn definitions of concepts from positive and negative instances.
  - It induces a **version space** - the set of all hypotheses that are *consistent* with the given positive examples.
  - The top-most concept is the most general. Concepts become more specific down the lattice, eventually reaching the positive examples.



  - Aim to find hypothesis general enough to cover positive examples but specific enough not to cover negative examples.

- **Generality**
  - We say h is more **general** than h' ($h \succcurlyeq h'$) iff $c(h', E) \subseteq c(h, E)$.
    - i.e. h covers all examples covered by h'.
    - In learning by entailment, $C \succcurlyeq D$ iff $C \vDash D$.
      - We say C **θ-subsumes** D iff $\exists\theta$ with $C\theta \subseteq D$.
      - If C θ-subsumes D, then $C \vDash D$ (but not converse!)
      - Unlike $\vDash$, subsumption is decidable and thus a pruning strategy.
  - If $c(h, \{e^-\})$, then, for any $g \succcurlyeq h$, we have $c(g, \{e^-\})$.
    - Here, an ILP search needs to **specialise** (prune parents).
  - If $\neg c(h, \{e^+\})$, then, for any $s \preccurlyeq h$, we have $\neg c(s, \{e^+\})$.
    - Here, an ILP search needs to **generalise** (prune children).
  - The lattice of clauses can be modified so that each node represents an equivalence class between clauses that θ-subsume each other.

- **ILP Learning Strategies**
  - Below, we make use of operators based on θ-subsumption.
  - **General to specific traversal** (Top-down)
    - Use a refinement or specialisation operator ρ.
      - Add a literal in the body of the clause.
      - Apply a substitution θ.
    - Start from most general clause and keep refining until no e^- is covered.
  - **Specific to general traversal** (Bottom-up)
    - **Plotkin's least general generalisation** (lgg) operator: derive the most specific clause that generalises two given clauses.

      (terms) $lgg(a, b) = X,\ lgg(f(X), g(Z)) = W$

      $lgg(f(a, b), f(c, c)) = f(X, Y)$

      (literals) $lgg(p(s_1, \ldots, s_n), p(t_1, \ldots, t_n)) = p(lgg(s_1, t_1), \ldots, lgg(s_n, t_n))$

      (clauses) $lgg(C_1, C_2)$

      $= \{lgg(l_1, l_2) \mid l_1 \in C_1, l_2 \in C_2 \wedge lgg(l_1, l_2) \text{ defined}\}$
    - Start from most specific clauses (positive examples) and keep finding lgg until most general clause not covering negative examples is found.

4 Combining Bottom-Up and Top-Down Search

- **Language bias**: a set of *mode declarations* that defines the language of the hypothesis being searched (and thus restricts the search space).
  - **Mode declarations** indicate the predicate that may appear in either the rule's head ($modeh(r, s)$) or body ($modeb(r, s)$).
    - r (**recall**) indicates how many times a predicate may appear.
      - By convention, $r = 1$ for *modeh* declarations.
      - $r = *$ indicates it can be used as many times as possible.
    - s (**scheme**) is a ground atom with placemarkers in the predicate.
      - Placemarkers: $+t$ (input), $-t$ (output), $\#t$ (constant)
      - Note: a variable can be an input variable if it is an output variable in a predicate before it, e.g. $A(X) \leftarrow B(X, Y), C(Y)$.

- **Inverse Entailment** property: $B \cup \{h\} \vDash e^+$ iff $B \cup \{\neg e^+\} \vDash \neg h$.
  - Equivalently useful: iff $B \cup \{\neg e^+\} \cup a \vDash [\ ]$.

- **Bottom Generalisation** (BG)
  - For a ground atom $e \in E^+$, theory B, and definite clause h,
  - The **bottom set** of B and e is: $Bot(B, e) = \{lg \mid B \cup \{\neg e\} \vDash \neg lg\}$.
  - We say h is **derivable by BG from** B **and** e iff $h \succcurlyeq_\theta Bot(B, e)$, i.e. $H\theta \subseteq Bot(B, e)$ for some θ.

- **PROGOL** Procedure
  1. Pick a (seed) positive example e^+.
  2. (**BOTTOMSET**)
     a. Compute the negated bottomset: $B \cup \{\neg e^+\delta\} \vDash \neg Bot(B, e^+)$.
        - Either through SLD resolution or finding $LHM(B \cup \{\neg e^+\delta\})$.
     b. Negate the result to get the bottom set $Bot(B, e^+)$.
     c. Find its immediate generalisation, the bottom clause $h_\perp$, by replacing its constants with unique variables.
  3. (**SEARCH**)
     a. Find the most general h with $h \succcurlyeq_\theta h_\perp$ that does not entail any negative examples through a top-down refinement process.
  4. Add h to B; remove covered e within E^+. Go to (1) if $E^+ \neq \emptyset$.

- **Observation Predicate Learning (OPL)**: learning predicates whose heads are the same as observed examples.
  - BG with SLD (PROGOL) can only solve OPL tasks.
    - This is because we may need to derive $\neg P(x)$ for some P, but this does not appear in any of the rule heads.
    - PROGOL5 solves this by including each rule's contrapositives in B.
    - This STARTSET is **incomplete** when a ground head atom needs to be used more than once in SLD derivation. This can be shown by deriving a failure with the negated head atom as a goal.

- **PROGOL5** Procedure
  1. Pick a (seed) positive example e^+.
  2. (**STARTSET**)
     a. Augment B with contrapositives of its rules and adding to it:
        - The skolemised body literals of e^+ if e^+ is a definite clause.
        - The negation of the (skolemised) head of e^+.
     b. For each $modeh(r, s(.))$ in the language bias M, start an SLD derivation of its predicate's negated form (i.e. $\leftarrow non\_s(.)$)
        - If successful, this step returns an atom $a = s\theta$. Set a to be the head of the hypothesis.
  3. (**BOTTOMSET**)
     a. With a as the head atom, derive body atoms b_i that unify with θ and satisfy *modeb* declarations *through SLD on* $B \cup \{\neg e^+\}$.
     b. The ground bottom clause is: $ground(h_\perp) = a \leftarrow b_1\theta, ..., b_n\theta$.
     c. Find its immediate generalisation, $h_\perp = s \leftarrow b_1, ..., b_n$
  4. (**SEARCH**)
     a. Find most compressed (least number of literals) hypothesis $h \in S_M$ with $h \succcurlyeq_\theta h_\perp$ without entailing any $e^- \in E^-$.
  5. Add h to B; remove covered e within E^+. Go to (1) if $E^+ \neq \emptyset$.

- **Kernel Set Subsumption** (KSS)
  - For a ground atom e, set of definite clauses B, and a set of ground definite clauses $ground(K) = k_1, ..., k_n$ with $k_i = a_i \leftarrow b_{i1}, ..., b_{im}$,
    - $ground(K)$ is a **ground Kernel Set** of B and e
      - iff $B \cup \{a_i \wedge ... \wedge a_n\} \vDash e$ and $B \cup \{\neg e\} \vDash b_{ij}$ for all pairs i, j.
    - The **Kernel Set** K of B and e is the immediate generalisation of $ground(K)$ compatible with the mode declarations.
    - A set of definite clauses H is **derivable by KSS** from B and e
      - iff $H \succcurlyeq_\theta K$ for some Kernel Set K of B and e.

➤ **HAIL** Algorithm

1. Select a (seed) positive example e^+.
2. (**Abduction**)
   a. Create an abutive task $\langle B, Ab, IC\rangle, O = \{e^+\}$, with IC empty, $Ab = \{h\theta \mid modeh(\_, h()) \in M\}$, i.e. all ground head atoms.
   b. Find an abductive solution $\Delta = \{a_1, ..., a_n\} \subseteq Ab$.
3. (**Deduction**)
   a. Derive ground instances of every $modeb(\_, b_{ij}) \in M$ through $B \cup \{\neg e^+\} \vdash b_{ij}\theta$ where θ is the grounding used in Δ.
   b. Construct a set of rules with heads as in Δ and body atoms as derived above. This forms the grounded Kernel Set.
   c. Find the immediate generalisation to yield the Kernel Set K.
4. (**Induction**)
   a. Find the most compressed h with $h \succcurlyeq_\theta K$.
      - Can do this by drawing out entire lattice (K is at the bottom) and traversing down until no negative examples covered.
      - Can use abduction: $\langle B \cup T, \{use()\}, \emptyset\rangle$, $O = e \wedge not\ e_1^- \wedge ... not\ e_n^-$.
        - For each $a_i \leftarrow b_{i1}, ..., b_{ik}$ in $K(B, e)$, $j \in [1, k]$ add to your T:

$$a_i \leftarrow use(i, 0), try(i, 1, X^{i1}), ..., try(i, k, X^{ik}).$$

$$try(i, j, X^{ij}) \leftarrow use(i, j), b_{ij}. \quad try(i, j, X^{ij}) \leftarrow not\ use(i, j).$$

5. Add h to B; remove covered e within E^+. Go to (1) if $E^+ \neq \emptyset$.

5 Top-Directed Abductive Learning

- Aim to use mode declarations to define a **meta-level theory** capturing the search space and compute inductive solutions from it.
- Allows computation of recursive logic programs, multiple clauses/hypotheses, and (TAL) learning from normal logic programs.
- **For below, we consider the mode M** with $m1 : modeh(1, p(+any))$, $m2 : modeb(*, q(+any, -any))$, $m3 : modeb(*, r(+any, \#const))$.

➤ **TopLog** Procedure

1. Construct a top theory T.
   a. For each $modeh$, e.g. $p(+any)$, add to your T:

$$p(X) \leftarrow \$body(X)$$
$$\$body(X) \leftarrow$$

   (multiple $modeh$ can lead to multiple starts)
   b. For each $modeb$, e.g. $q(+any, -any)$, $r(+any, \#const)$), find $\$body$ predicates which can be unified and add to your T:

$$\$body(X) \leftarrow q(X, Y), \$body(Y)$$
$$\$body(X) \leftarrow r(X, C), const(C), \$body(X)$$

2. For each positive example e^+, construct an SLD derivation of e^+ from $B \cup T$. Each derivation corresponds to a hypothesis.
3. Find the hypothesis with best coverage, maximising:

$$score(h) = \#entailed(E^+) - \#entailed(E^-)$$

- **Monotonicity** assumption: $B \cup H_1 \vDash e_1 \implies B \cup H_1 \cup H_2 \vDash e_1$. This holds for definite LPs, but, in general, not for normal LPs.

➤ **TAL** Procedure

1. Encode modes as $\langle m_{label}, [consts], [inputs]\rangle$, e.g.

$$p(X) \leftarrow q(X, Y), r(Y, a)$$
$$\langle m1, [], []\rangle, \langle m2, [], [1]\rangle, \langle m3, [a], [2]\rangle$$

2. Construct a top-theory T_M, using aux predicates: $link$ (to number inputs), $append$ (to append to lists), $rule$ (to mark rules), $body$, e.g.

$$p(V1) \leftarrow \mathbf{body}([V1], [\langle m1, [], []\rangle]).$$
$$\mathbf{body}(InputSoFar, RuleSoFar) \leftarrow \mathbf{rule}(RuleSoFar).$$
$$\mathbf{body}(InputSoFar, RuleSoFar) \leftarrow$$
$$\mathbf{link}([V1], InputSoFar, Links), q(V1, V2),$$
$$\mathbf{append}(RuleSoFar, [\langle m2, [], Links\rangle], NewRule),$$
$$\mathbf{append}(InputSoFar, [V2], NewInputs),$$
$$\mathbf{body}(NewInputs, NewRule).$$
$$\mathbf{body}(InputSoFar, RuleSoFar) \leftarrow$$
$$\mathbf{link}([V2], InputSoFar, Links), r(V2, A),$$
$$\mathbf{append}(RuleSoFar, [\langle m3, [A], Links\rangle], NewRule),$$
$$\mathbf{append}(InputSoFar, [], NewInputs),$$
$$\mathbf{body}(NewInputs, NewRule).$$

3. Solve the abductive task $\langle B \cup T_M,\ \{rule(.)\},\ IC = \emptyset\rangle$, $O = E$. Starting goal is $\leftarrow e_1^+, ..., e_n^+, not\ e_1^-, ..., not\ e_m^-$. In the derivation, every aux predicate before $body$ can be removed one at a time.
4. Decode the rules in the abductive solution Δ.

6 Stable Model Semantics

- Differences with previous paradigms:
  - In ASP, the order of body literals *does not matter*, but rules need to be **safe**: all variables of a rule R must occur in $body^+(R)$.
  - Normal LPs may have a *non-unique LHM* - it may even be unsupported.
- Finding $ground(P)$, the **relevant grounding** of a program P.
  1. Add all the ground facts of P.
  2. Add all ground instances of rules whose grounded *positive* body atoms all appear as a head of some rule in $ground(P)$.
  3. Repeat (1) and (2) until no additional rule is added to $ground(P)$.
- Finding P^X, the **reduct** of (a ground) P with respect to X.
  1. Remove all rules R whose $body^-(R)$ contains some $x \in X$.
  2. Remove $body^-(R)$ for all remaining rules in R.
- An interpretation X is a **stable model/answer set** of a normal LP P iff $X = LHM((ground(P))^X)$.
- An atom A is **bravely entailed** ($P \vDash_b A$) by a normal LP P if it is true in **at least one stable model** of P. Whereas, it is **cautiously entailed** ($P \vDash_c A$) if it is true in **all stable models** of P.

7 Answer Set Programming

- **Constraints** ($:- b_1, ..., b_m, not\ c_1, ..., not\ c_n$) filter out unwanted answer sets. When computing the reduct, the empty head is replaced with $\perp$.
- **Choice rules**: ($a\{h_1, ..., h_n\}b :- ...$) allow the use of *disjunctions* in the rule head. When computing the reduct, if the aggregate A, wrt X, is:
  - **Not satisfied**: remove the head (i.e. make the rule into a constraint).
  - **Satisfied**: copy the rule for each satisfied atom in A with it as the head.
- **Opimisation statements**: ($\#minimize[\ a_1 = w_1, ..., a_n = w_n\ ]$) decide which solutions are more optimal than others. Usually, $w_i = \#vars(a_i)$.

8 Abduction in ASP and Cautious/Brave Induction

- An abductive task $\langle B, Ab, IC\rangle$, O can be represented in ASP by:
  - Listing out all rules in B.
  - Creating one big choice rule for all the (atom) abducibles in Ab.
  - Set those in IC as constraints, and set the negation of O as a constraint.
- **Cautious induction** (ILP Task): given a background knowledge B, examples $\langle E^+, E^-\rangle$, find a hypothesis H such that:
  - $B \cup H$ is satisfiable (has at least one answer set)
  - For **all** answer sets A of $B \cup H$, all e^+ are covered and no e^- are covered.
- **Brave induction** (ILP Task): as above, but there only needs to be **at least one** answer set A that covers all e^+ and does not cover any e^-.
  - Weaker than above but **can not learn a constraint as a hypothesis**.

➤ **ASPAL: ASP Encoding** Procedure

1. Add the background knowledge B.
2. Create and add the **skeleton rules** from the mode declarations.
   - Add all possible rules that can be constructed from the mode declarations with constant placemarkers (e.g. $C1$) for constants, satisfying:
     - L_{max}: max # of literals allowed to appear in the rule body (excluding those for types); V_{max}: max # of variables per rule.
   - At the end of each rule, add an atom $rule(i, C1, ..., Cn)$ to distinguish it as the ith rule with constant placemarkers $C1, ..., Cn$
3. Generate **hypotheses** from the skeleton rules.
   a. Add a choice rule with all ground instances of the *rule* predicates.
   b. Add a minimisation statement that adds weights to each atom in the choice rule above, based on the number of atoms in the rule (not including those for asserting types nor the *rule* atom).
4. Add $goal :- e_1^+, ..., e_n^+, not\ e_1^-, ..., not\ e_m^-.$, and $:- not\ goal$.

9 Learning From Answer Sets

- Motivation: express both brave and cautious induction within a single learning task by having **(partial) answer sets as examples** rather than atoms. Hence, learning from **answer sets**.
- A **partial interpretation** $e = \langle e^{inc}, e^{exc}\rangle$ is a pair of sets of atoms - the *inclusions* and the *exclusions*.
  - An interpretation I **extends** e iff $e^{inc} \subseteq I$ and $e^{exc} \cap I = \emptyset$.
- **LAS** (ILP Task): given $\langle B, S_M, E^+, E^-\rangle$ with ASP program B, partial interpretations E^+, E^-, and hypothesis space S_M.
  Find a hypothesis H with $H \subseteq S_M$ such that:
  - $\forall e^+ \in E^+ : \exists A \in AnswerSets(B \cup H)$ st A extends e^+.
  - $\forall e^- \in E^- : \nexists A \in AnswerSets(B \cup H)$ st A extends e^-.
- Some examples for interpretting:

	$\langle\{v(C1, h), v(C2, h)\},\ \emptyset\rangle$	$\langle\emptyset,\ \{v(C1, h), v(C2, h)\}\rangle$
In E^+	at least one AS contains both	at least one AS contains neither
In E^-	no AS contains both	all ASs must contain at least one of the two

- Relation to brave and cautious induction:
  - $ILP_{brave}\langle B, E^+, E^-\rangle \mapsto ILP_{LAS}\langle B, \{\langle E^+, E^-\rangle\}, \emptyset\rangle$
  - $ILP_{cautious}\langle B, E^+, E^-\rangle$
    $\mapsto ILP_{LAS}\langle B,\ \{\langle\emptyset, \emptyset\rangle\},\ \{\langle\emptyset, e_i^+\rangle \mid e_i^+ \in E^+\} \cup \{\langle e_i^-, \emptyset\rangle \mid e_i^- \in E^-\}\rangle$

10 Probabilistic Logic Programming

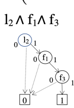
- A probabilistic logic **program database** is made up of a set of (mutually independent Boolean) **probabilistic facts** F and a set of **rules** R, where no fact unifies with any of its rule heads (the *disjoint condition*).
- **(Herbrand) interpretations as possible worlds**
  - A **possible world** ω is a vector $\langle x_1, ..., x_n\rangle$ with x_i as an outcome of F_i.
  - In a sense, the prob. distributions of the given facts determine a prob. distribution over the *set of possible worlds* W_F (logic programs Π_i).
- Finding the **probability of a possible world**
  - Define $(F_i, 1)$ as atom F_i being selected and $(F_i, 0)$ if not.
  - Given a set of probabilistic atoms $F = \{P_1 :: F_1, ..., P_n :: F_n\}$ and a composite choice $k = \{(F_1, x_1), ..., (F_n, x_n)\}$ for $x_i \in \{0, 1\}$, we have

$$P(k) = \prod_{(F_i,1)\in k} P_i \times \prod_{(F_i,0)\in k} (1 - P_i)$$

  - We also have **probability of a formula/atom**: $P(G) = \sum_{\omega_i \vDash G} P(\omega_i)$.

➤ **ProbLog: Efficient inference of success query with BDDs**

1. Construct the entire SLD proof tree of a given query q wrt to the probabilistic logic program T. Take the disjunction of all the conjuncts corresponding to each successful branch. (Order conjuncts consistently).
2. Convert found DNF formula into a **Binary Decision Diagram** (BDD).
   - Convert each disjunct, then compose them iteratively.
   - BDD of each conjunction is just a tree with each node connected to zero and the next literal (or 1-terminal).
   - Compose disjuncts by treating each group as a single literal



3. Find probability $P(n)$ of root BDD node n (corresponding to fact f_n).
   a. If n is 1-terminal (or 0-terminal), then return 1 (or 0).
   b. Else return $p_n \cdot P(child_1) + (1 - p_n) \cdot P(child_0)$, where $p_n :: f_n$.