CS 188 Introduction to Artificial Intelligence
Fall 2018 Note 11

These lecture notes are written by Henry Zhu. All figures are from *Artificial Intelligence: A Modern Approach*.

A Knowledge Based Agent

Until now, we have essentially considered the task of learning to be one of association. To illustrate this, imagine a dangerous world filled with lava, the only respite a far away oasis. We would like our agent to be able to safely navigate from its current position to the oasis. In reinforcement learning, we assume that the only guidance we can give is a reward function which will try to nudge the agent in the right direction, like a game of 'hot or cold'. As the agent explores and collects more observations about the world, it gradually learns to associate certain actions with positive future reward, and certain ones with undesirable, scalding death. To this end, it may learn to recognize certain cues from the world and act accordingly, for example, if it feels the air getting hot it should turn the other way and if it feels a cool breeze it should keep going.

However, we might consider an alternative strategy. Instead let's tell the agent some facts about the world and allow it to reason about what to do, based on the information at hand. If we told the agent that air gets hot and hazy around pits of lava, or crisp and clean around bodies of water, then it could reasonably infer what areas of the landscape are dangerous or welcome based on its readings of the atmosphere. This alternative type of agent is known as a **knowledge based agent**. Such an agent maintains a **knowledge base**, which is a collection of logical **sentences**, that encodes what we have told the agent and what it has observed, and is able to perform **logical inference** in order to deduce further information from this.

The Language of Logic

Just as with any other language, logic sentences are written in a special **syntax**. Every logical sentence is code for a **proposition** about a world that may or may not be true. For example the sentence "the floor is lava" may be true in our agent's world, but probably not true in ours. We can construct complex sentences by stringing together simpler ones with **logical connectives**, to create sentences like "you can see all of campus from the Big C *and* hiking is a healthy break from studying". There are five logical connectives in the language:

- $\neg$, **not**: $\neg P$ is true *iff* P is false. The atomic sentences P and $\neg P$ are referred to as **literals**.
- $\wedge$, **and**: $A \wedge B$ is true *iff* both A is true and B is true. An 'and' sentence is known as a **conjunction** and its component propositions the **conjuncts**.
- $\vee$, **or**: $A \vee B$ is true *iff* either A is true or B is true. An 'or' sentence is known as a **disjunction** and its component propositions the **disjuncts**.
- $\Rightarrow$, **implication**: $A \Rightarrow B$ is true unless A is true and B is false.
- $\Leftrightarrow$ or $\equiv$, **biconditional**: $A \Leftrightarrow B$ ($A \equiv B$) is true *iff* either both A and B are true or both are false.

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Figure 7.8 Truth tables for the five logical connectives. To use the table to compute, for example, the value of $P \vee Q$ when P is true and Q is false, first look on the left for the row where P is *true* and Q is *false* (the third row). Then look in that row under the $P \vee Q$ column to see the result: *true*.

Below are some useful logical equivalences, which can be used for simplifying sentences to forms that are easier to work and reason with.

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Figure 7.11 Standard logical equivalences. The symbols α, β, and γ stand for arbitrary sentences of propositional logic.

One particularly useful form is the **conjunctive normal form** or **CNF** which is is a conjunction of clauses, each of which a disjunction of literals. It has the general form $(P_1 \vee \cdots \vee P_i) \wedge \cdots \wedge (P_j \vee \cdots \vee P_n)$. As we'll see, a sentence in this form is amenable to certain analyses, and importantly, every logical sentence has a logically equivalent conjunctive normal form. This means that we can formulate all the information contained in our knowledge base (which is just a conjunction of sentences) as one large conjunctive normal form.

Like other languages, logic has multiple dialects; we will introduce two. The first, **propositional logic**, is written in sentences composed of **proposition symbols**, possibly joined by logical connectives. Each proposition symbol stands for an atomic proposition about the world. A **model** is an assignment of true or false to all the proposition symbols, which we might think of as a "possible world". For example, if we had the propositions A = "today it rained" and B = "I forgot my umbrella" then the possible models (or "worlds") are:

1. {A=true, B=true} ("Today it rained and I forgot my umbrella.")
2. {A=true, B=false} ("Today it rained and I didn't forget my umbrella.")
3. {A=false, B=true} ("Today it didn't rain and I forgot my umbrella.")
4. {A=false, B=false} ("Today it didn't rain and I did not forget my umbrella.")

In general, for N symbols, there are 2^N possible models. We say a sentence is **valid** if it is true in all of these models (e.g. the sentence *True*), **satisfiable** if there is at least one model in which it is true, and **unsatisfiable** if it is not true in any models. For example, the sentence $A \wedge B$ is satisfiable because it is true in model 1, but not valid since it is false in models 2, 3, 4.

The second, **first-order logic**, is more expressive, and uses objects as its basic components. With first-order logic we can describe relationships between objects, as well as apply functions to them. Each object is represented by a **constant symbol**, each relationship by a **predicate symbol**, and each function by a **function symbol**. Atomic sentences in first-order logic are descriptions of relationships between objects, and are true if the relationship holds. Complex sentences of first order logic are analogous to those in propositional logic and are atomic sentences sewn together by logical connectives.

Naturally we would like ways to describe entire collections of objects. For this we use **quantifiers**. The **universal quantifier** $\forall$, has the meaning "for all" and the **existential quantifier** $\exists$, has the meaning "there exists". For example, if the set of objects in our world is debates, $\forall a\ TwoSides(a)$ could be translated as "there are two sides to every debate", and if the set of objects in our world is people, $\forall x, \exists y,\ SoulMate(x,y)$ as "for all people, there is someone out there who is their soulmate", a.k.a. "everyone has a soulmate". The anonymous **variables** a, x, y are standins for objects, and can be **substituted** for actual objects, for example, substituting {x/Laura} into our second example would result in a statement that "there is someone out there for Laura". The universal and existential quantifiers are convenient ways to express a conjunction and disjunction over all objects, and hence also obey De Morgan's laws (note the analogous relationship between conjunctions and disjunctions):

$$\begin{array}{lcl} \forall x\ \neg P & \equiv & \neg\exists x\ P \\ \neg\forall x\ P & \equiv & \exists x\ \neg P \\ \forall x\ P & \equiv & \neg\exists x\ \neg P \\ \exists x\ P & \equiv & \neg\forall x\ \neg P \end{array} \qquad \begin{array}{lcl} \neg(P \vee Q) & \equiv & \neg P \wedge \neg Q \\ \neg(P \wedge Q) & \equiv & \neg P \vee \neg Q \\ P \wedge Q & \equiv & \neg(\neg P \vee \neg Q) \\ P \vee Q & \equiv & \neg(\neg P \wedge \neg Q) \end{array}$$

Finally, we use the **equality symbol** to signify that two symbols refer to the same object. For example, the incredible sentence $(Wife(Einstein) = FirstCousin(Einstein) \wedge Wife(Einstein) = SecondCousin(Einstein))$ is true!

Unlike with propositional logic, where a model was an assignment of true or false to all proposition symbols, a model in first-order logic is a mapping of all constant symbols to objects, predicate symbols to relations between objects, and function symbols to functions of objects. A sentence is true under a model if the relations described by the sentence are true under the mapping. While the number of models of a propositional logical system is always finite, there may be an infinite number of models of a first order logical system if the number of objects is unconstrained.

These two dialects of logic allow us to describe and think about the world in different ways. With propositional logic, we model our world as a set of attributes, that are true or false. Under this assumption, we can represent a possible world as a vector, a 1 or 0 for every attribute. This binary view of the world is what is known as a **factored representation**, which we used when we were solving CSP's. With first-order logic, our world consists of objects that relate to one another. The second object-oriented view of the world is known as a **structured representation**, is in many ways more expressive and is more closely aligned with the language we naturally use to speak about the world.

Propositional Logical Inference

Logic is useful, and powerful, because it grants the ability to draw new conclusions from what we already know. To define the problem of inference we first need to define some terminology.

We say that a sentence A **entails** another sentence B if in all models that A is true, B is as well, and we represent this relationship as $A \models B$. Note that if $A \models B$ then the models of A are a subset of the models of B. The inference problem can be formulated as figuring out whether $KB \models q$, where KB is our knowledge base of logical sentences, and q is some query. For example, if Simin has avowed to never set foot in Crossroads again, we could infer that we will not find her when looking for friends to sit with for dinner.

We draw on two useful theorems to show entailment:

i.) ($A \models B$ *iff* $A \Rightarrow B$ is valid).

Proving entailment by showing that $A \Rightarrow B$ is valid is known as a **direct proof**.

ii.) ($A \models B$ *iff* $A \wedge \neg B$ is unsatisfiable).

Proving entailment by showing that $A \wedge \neg B$ is unsatisfiable is known as a **proof by contradiction**.

Model Checking

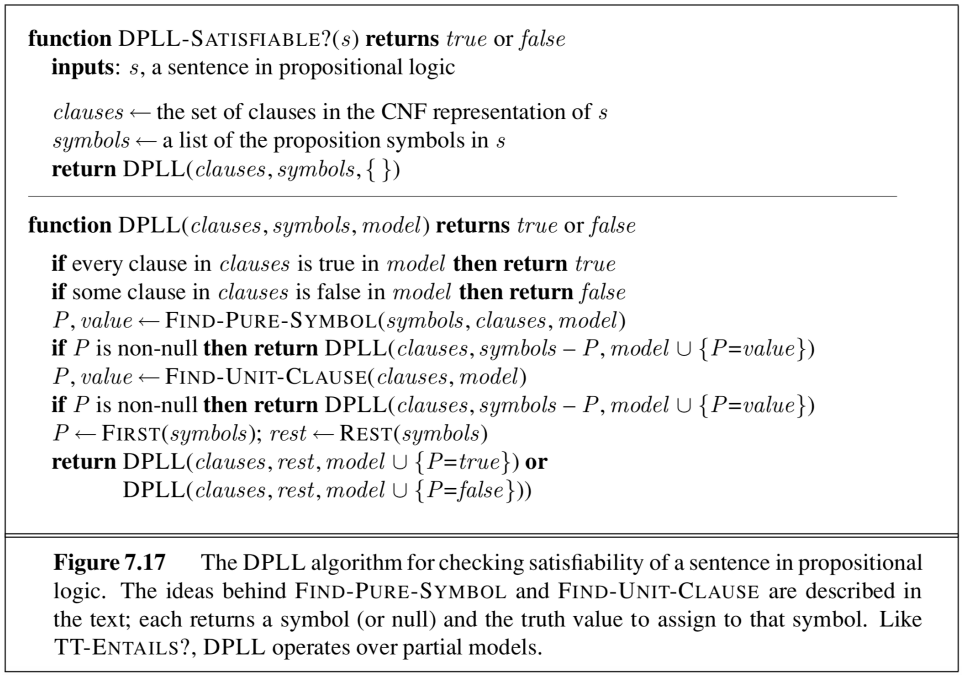
One simple algorithm for checking whether $KB \models q$ is to enumerate all possible models, and to check if in all the ones in which KB is true if q is true as well. This approach is known as **model checking**. For a propositional logical system if there are N symbols, there are 2^N models to check, and hence the time complexity of this algorithm is $O(2^N)$, while in first-order logic, the number of models is infinite. In fact the problem of propositional entailment is known to be co-NP-complete. While the worst case runtime will inevitably be an exponential function of the size of the problem, there are algorithms that can in practice terminate much more quickly. We will discuss two model checking algorithms for propositional logic.

The first, proposed by Davis, Putnam, Logemann, and Loveland (which we will call the **DPLL algorithm**) is essentially a depth-first, backtracking search over possible models with three tricks to reduce excessive backtracking. This algorithm aims solve the satisfiability problem, i.e. given a sentence, find a working assignment to all the symbols. As we mentioned, the problem of entailment can be reduced to one of satisfiability (show that $A \wedge \neg B$ is not satisfiable), and specifically DPLL takes in a problem in CNF. Satisfiability can be formulated as a constraint satisfaction problem as follows: let the variables (nodes) be the symbols, and the constraints the logical constraints imposed by the CNF. Then DPLL will procede just like backtracking search: continue assigning symbols truth values until either a satisfying model is found or a symbol cannot be assigned without violating a logical constraint, at which point the algorithm will backtrack to the last working assignment. However, DPLL makes three improvements over simple backtracking search:

1. **Early Termination**: A clause is true if any of the symbols are true. Therefore the sentence could be known to be true even before all symbols are assigned. Also, a sentence is false if any single clause is false. Early checking of whether the whole sentence can be judged true or false before all variables are assigned can prevent unnecessary meandering down subtrees.

2. **Pure Symbols**: A pure symbol is a symbol that only shows up in its positive or negative form throughout the entire sentence. Pure symbols can immediately be assigned true or false. e.g. A is the only pure symbol in $(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee A)$. , and can immediately be assigned true, reducing the satisfying problem to one of just finding a satisfying assignment of $(\neg B \vee C)$.

3. **Unit Clauses**: A unit clause is a clause with just one literal or a disjunction with one literal and many falses. In a unit clause, we can immediately assign a value to the literal, since there is only one valid assignment. e.g. B must be true for the unit clause $(B \vee false \vee \cdots \vee false)$ to be true.



function DPLL-SATISFIABLE?(s) **returns** *true* or *false*
  inputs: s, a sentence in propositional logic

  $clauses \leftarrow$ the set of clauses in the CNF representation of s
  $symbols \leftarrow$ a list of the proposition symbols in s
  return DPLL($clauses, symbols, \{\}$)

function DPLL($clauses, symbols, model$) **returns** *true* or *false*

  if every clause in $clauses$ is true in $model$ **then return** *true*
  if some clause in $clauses$ is false in $model$ **then return** *false*
  $P, value \leftarrow$ FIND-PURE-SYMBOL($symbols, clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P$=$value\}$)
  $P, value \leftarrow$ FIND-UNIT-CLAUSE($clauses, model$)
  if P is non-null **then return** DPLL($clauses, symbols - P, model \cup \{P$=$value\}$)
  $P \leftarrow$ FIRST($symbols$); $rest \leftarrow$ REST($symbols$)
  return DPLL($clauses, rest, model \cup \{P$=$true\}$) **or**
    DPLL($clauses, rest, model \cup \{P$=$false\}$))

Figure 7.17 The DPLL algorithm for checking satisfiability of a sentence in propositional logic. The ideas behind FIND-PURE-SYMBOL and FIND-UNIT-CLAUSE are described in the text; each returns a symbol (or null) and the truth value to assign to that symbol. Like TT-ENTAILS?, DPLL operates over partial models.

The second approach similarly formulates the entailment problem as a CSP, and finds a solution using local search. This algorithm is known as **Walk-SAT** and involves randomly initializing all symbols and then iteratively choosing an unsatisfied clause and choosing a symbol to "flip". The algorithm can either choose the symbol that will result in the most number of satisfied clauses, or choose a symbol at random to avoid getting stuck in local minima.

Theorem Proving

An alternate approach is to apply rules of inference to KB in order to prove that $KB \models q$. For example, if our knowledge base contains A and $A \Rightarrow B$ then we can infer B (this rule is known as *ModusPonens*). The two previously mentioned algorithms use the fact ii.) by writing $A \wedge \neg B$ in CNF and show that it is either satisfiable or not.

We could also prove entailment using two rules of inference:

1. If our knowledge base contains A and $A \Rightarrow B$ we can infer B (**Modus Ponens**)
2. If our knowledge base contains $A \wedge B$ we can infer A or B (**And-Elimination**).
3. If our knowledge base contains A and B we can infer $A \wedge B$ (**Resolution**).

The last rule forms the basis of the **resolution algorithm** which iteratively applies it to the knowledge base and to the newly inferred sentences until either q is inferred, in which case we have shown that $KB \models q$, or there is nothing left to infer, in which case $KB \not\models q$. Although this algorithm is both sound (the answer will

be correct) and complete (the answer will be found) it runs in worst case time that is exponential in the size of the knowledge base.

However, in the special case that our knowledge base is consists solely of literals and implications: $(P_1 \wedge \cdots \wedge P_n \Rightarrow Q) \equiv (\neg P_1 \vee \cdots \vee \neg P_2 \vee Q)$, we can prove entailment in time linear to the size of the knowledge base. One algorithm, **forward chaining** iterates through every implication statement in which the LHS is known to be true, adding the RHS to the list of known facts. This is repeated until q is implied, or nothing more can be inferred. Conversely, **backward chaining** finds an implication statement that implies q and recursively tries prove each conjunct on the RHS, the base case being if the conjunct is already in the knowledge base. We can think of forward chaining as an example of *data-driven reasoning* and backward chaining an example of *goal-oriented reasoning*.

First Order Logical Inference

With first order logic we formulate inference exactly the same way. We'd like to find out if $KB \models q$, that is if q is true in all models under which KB is true. One approach to finding a solution is **propositionalization** or translating the problem into propositional logic so that it can be solved with techniques we have already laid out. Each universal (existential) quantifier sentence can be converted to a conjunction (disjunction) of sentences with a clause for each possible object that could be substituted in for the variable. Then, using a SAT solver, like DPLL or Walk-SAT, (un)satisfiability of $(KB \wedge \neg q)$.

One problem with this approach is there are an infinite number of substitutions that we could make, since there is no limit to how many times we can apply a function to a symbol. For example, we can nest the function $Classmate(\cdots Classmate(Classmate(Austen))\cdots)$ as many times as we'd like, until we reference the whole school. Luckily, a theorem proved by Jacques Herbrand (1930) tells us that if a sentence is entailed by a knowledge base that there is a proof involving just a *finite* subset of the propositionalized knowledge base. Therefore, we can try iterating through finite subsets, specifically searching via iterative deepening through nested function applications, i.e. first search through substitutions with constant symbols, then substitutions with $Classmate(Austen)$, then substitutions with $Classmate(Classmate(Austen))$, ...

Another approach is to directly do inference with first order logic. Given $(\forall x\ HasAbsolutePower(x) \wedge Person(x) \Rightarrow Corrupt(x)) \wedge Person(John) \wedge HasAbsolutePower(John)$ ("absolute power corrupts absolutely") we can infer $Corrupt(John)$ by substituting x for $John$, $\{x/John\}$. This rule is known as **generalized Modus Ponens**. The forward chaining algorithm for first order logic repeatedly applies generalized Modus Ponens and substitution to infer q or show that it cannot be inferred. Also just like with propositional logic, the backward chaining algorithm recursively attempts to infer the premises (RHS) of $O_1 \wedge \cdots \wedge O_2 \Rightarrow q$ through application of generalized Modus Ponens and substitution.

Let's look at any example. Let $B = Price(BubbleTea)$, $H = Price(MealAtHome)$, and $T = Price(Takeout)$. Given the axioms below, prove that $H < B + T$. Note that we freely rename variables to avoid substituting different values for the same variable.

Axioms

(1) $0 \leq B$

(2) $H \leq T$

(3) $\forall x,\ x \leq x$

(4) $\forall x,\ x \leq x + 0$

(5) $\forall x,\ x + 0 \leq x$

(6) $\forall x, y,\ x + y \leq y + x$

(7) $\forall w, x, y, z,\ w \leq y \wedge x \leq z \Rightarrow w + x \leq y + z$

(8) $\forall x, y, z,\ x \leq y \wedge y \leq z \Rightarrow x \leq z$

Forward Chaining Proof

i. From (7) {w/0, y/B, x/H, z/T} infer that $0+H \leq B+T$.

ii. From (6) {y1/0, x1/H} infer that $H+0 \leq 0+H$.

iii. From (4) {x2/H} infer that $H \leq H+0$.

iv. From (8), (ii), (iii) {x3/H, y3/H+0, z3/0+H} infer that $H \leq 0+H$.

v. From (8), (i), (iv) {x4/H, y4/0+H, z4/B+T} infer that $H \leq B+T$.

Backward Chaining Proof

1. Goal: $H \leq B+T$. From (8) and {x/H, z/B+T} derive two subgoals: $H \leq y1$, $y1 \leq B+T$.
   (a) Goal: $H \leq y1$. Resolve with (4) and substitution {y1/H+0}.
   (b) Goal: $H+0 \leq B+T$. From (8) and {x2/H+0, z2/B+T} derive two subgoals: $H+0 \leq y2$, $y2 \leq B+T$.
      i. Goal: $H+0 \leq y2$. Resolve with (6) and substitution {y2/0+H, x3/H, y3/0}.
      ii. Goal: $0+H \leq B+T$. From (7) and substitution {w4/0, x4/H, y4/B, z4/T} derive two subgoals: $0 \leq B$, $H \leq T$.
         A. Goal: $0 \leq B$. Resolve with (1).
         B. Goal: $H \leq T$. Resolve with (2).

Logical Agents

Now that we understand how to formulate what we know and how to reason with it, we will talk about how to incorporate the power of deduction into our agents. One obvious ability an agent should have is the ability to figure out what state it is in, based on a history of observations and what it knows about the world (**state-estimation**). For example, if we told the agent that the air starts to shimmer near pools of lava and it observed that the air right before it is shimmering, it could infer that danger is nearby. In order to incorporate its past observations into an estimate of where it currently is, an agent will need to have a notion of time, and transitions between states. We call state attributes that vary with time **fluents** and write a fluent with an index for time, e.g. Hot^t = the air is hot at time t. Fluents encompass time-dependent parts of the state and actions at each time step and should remain constant over time, unless an action somehow disturbs its value. To represent this fact we can use the general form of the **successor-state axiom**

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)$$

In our world, the transition could be formulated as $Hot^{t+1} \Leftrightarrow StepCloseToLava^t \vee (Hot^t \wedge \neg StepAwayFromLava^t)$.

Having written out the rules of the world in logic, we can now formulate *planning as satisfiability*. Simply construct a sentence including information about the initial state, the transitions (successor-state axioms), and the goal (e.g. $InOasis^T \wedge Alive^T$ encodes the objective of surviving and ending up in the oasis by time T). If the rules of the world have been properly formulated, then finding a satisfying assignment to all the variables will allow us to extract a sequence of actions that will carry the agent to the goal.