Reasoning Algorithms in Artificial Intelligence

Sicun Gao
University of California, San Diego

Contents

Part I

Propositional Reasoning

Chapter 1

Propositional Logic

We write the language of propositional logic as $\mathcal{L}_0$.

1.1 Syntax

Definition 1.1.1 (Syntax of $\mathcal{L}_0$)**.** Let P be a set of *propositional variables*:

$$P := \{p_i \mid i \in \mathbb{N}\}.$$

The language $\mathcal{L}_0$ is defined as the minimal set that satisfies the following equation:

$$\mathcal{L}_0 = P \cup \{\neg p \mid p \in \mathcal{L}_0\} \cup \{p_1 \wedge p_2 \mid p_1, p_2 \in \mathcal{L}_0\}.$$

Exercise 1.1.1. Construct a bijection between $\mathcal{L}_0$ and $\mathbb{N}$.

Remark 1.1.1. To be extremely pedantic, $\mathcal{L}_0$ should be parameterized by the choice of the base set P, i.e., $\mathcal{L}_0(P)$. But just like how we normally use x, y, z to represent numerical variables, it is standard to write propositional variables as p, q, r etc.

Remark 1.1.2. Equivalently, we can define $\mathcal{L}_0$ as the minimal set that contains any formula φ of the following form:

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \varphi$$

where $p \in P$.

Exercise 1.1.2. Prove the equivalence between the two definitions.

Definition 1.1.2 (Disjunction and Implication)**.** Let $\varphi_1, \varphi_2 \in \mathcal{L}_0$ be arbitrary. Define:

$$\begin{aligned} \varphi_1 \vee \varphi_2 &:= \neg(\neg\varphi_1 \wedge \neg\varphi_2), \\ \varphi_1 \to \varphi_2 &:= \neg\varphi_1 \vee \varphi_2. \end{aligned}$$

Remark 1.1.3. This definition of implication was *invented* by Frege.

Exercise 1.1.3. What is a reasonable definition of the precendence order of the connectives?

Exercise 1.1.4. What are the other subsets of $\{\neg, \wedge, \vee, \to\}$ that can define all other connectives?

1.2 Semantics

Definition 1.2.1 (Interpretation)**.** An interpretation, or truth assignment, of $\mathcal{L}_0$ is any function $\sigma : P \to \{0, 1\}$.

Definition 1.2.2 (Evaluation)**.** Let σ be an interpretation of $\mathcal{L}_0$. The evaluation function

$$\widehat{\sigma} : \mathcal{L}_0 \to \{0, 1\}$$

is defined to satisfy the following conditions:

- For any $p \in P$, $\widehat{\sigma}(p) = \sigma(p)$.
- For any $\varphi \in \mathcal{L}_0$, $\widehat{\sigma}(\neg\varphi) = 1 - \widehat{\sigma}(\varphi)$.
- For any $\varphi_1, \varphi_2 \in \mathcal{L}_0$, $\widehat{\sigma}(\varphi_1 \wedge \varphi_2) = \widehat{\sigma}(\varphi_1) \cdot \widehat{\sigma}(\varphi_2)$.

Exercise 1.2.1. Prove the existence and uniqueness of this $\widehat{\sigma}$.

Remark 1.2.1. Because of the simplicity and uniqueness of the evaluation function, we often use "interpretation" and "evaluation" interchangeably, and write both as σ.

Remark 1.2.2. It is easy to see that such a $\widehat{\sigma}$ also satisfies that for any $\varphi_1, \varphi_2 \in \mathcal{L}_0$,

$$\widehat{\sigma}(\varphi_1 \vee \varphi_2) = \widehat{\sigma}(\varphi_1) + \widehat{\sigma}(\varphi_2) - \widehat{\sigma}(\varphi_1) \cdot \widehat{\sigma}(\varphi_2)$$

$$\widehat{\sigma}(\varphi_1 \to \varphi_2) = \widehat{\sigma}(\varphi_1) \cdot \widehat{\sigma}(\varphi_2) - \widehat{\sigma}(\varphi_1) + 1.$$

Exercise 1.2.2. Write down the truth table (the list of all possible evaluation functions) of the following formulas:

- $(p \to q) \to (\neg q \to \neg p)$
- $p \to (q \to r) \to ((p \to q) \to (p \to r))$

Definition 1.2.3 (Satisfiability)**.** We say $\varphi \in \mathcal{L}_0$ is satisfiable, if there exists an interpretation σ such that $\sigma(\varphi) = 1$. Otherwise we say it is unsatisfiable.

Definition 1.2.4 (Validity)**.** $\varphi \in \mathcal{L}_0$ is valid, if for any interpretation σ, $\sigma(\varphi) = 1$.

Proposition 1.2.1. φ is valid iff $\neg\varphi$ is unsatisfiable.

Example 1.2.1. The following formulas are valid:

- $p \to (q \to p)$
- $p \to (q \to r) \to ((p \to q) \to (p \to r))$

Definition 1.2.5 (The SAT Problem)**.** Given any $\varphi \in \mathcal{L}_0$, decide if φ is satisfiable. The dual problem of deciding whether φ is called the validity checking problem.

Remark 1.2.3. So far we have deliberately avoided giving any "logical" interpretation of the symbols "$\wedge, \vee, \to$." Indeed, they can be considered simply as convenient notations of certain functions defined on the value space of the propositional variables. However, it is easier to remember if we use our intuition about their interpretations as the logical "and, or, implies" respectively.

1.3 Complexity

Theorem 1.3.1. Satisfiability in $\mathcal{L}_0$ is NP-complete.

Corollary 1.3.1. Validity in $\mathcal{L}_0$ is co-NP-complete.

Corollary 1.3.2. Solving nonlinear equations over $\mathbb{F}_2$ (finite field of size 2) is NP-complete.

Exercise 1.3.1. Construct a bijection between $\mathcal{L}_0$ and polynomials over $\mathbb{F}_2$.

1.4 Application Examples

1.4.1 Discrete Motion Planning

(will insert a picture)

Consider the problem of moving a robot between to points in a cluttered environment.

1.4.2 Sudoku

Chapter 2

Boolean Satisfiability

2.1 CNFs and DNFs

Definition 2.1.1 (Conjunctive Normal Form)**.** We say $\varphi \in \mathcal{L}_0$ is in Conjunctive Normal Form (CNF), if it is of the form:

$$\varphi := \bigwedge_i (l_{i1} \vee \cdots \vee l_{ik_i})$$

where $l_{ij} = p_{ij}$ or $l_{ij} = \neg p_{ij}$ for some $p_{ij} \in P(\mathcal{L}_0)$. Here, each l_{ij} is called a *literal*.

Theorem 2.1.1. There exists an algorithm $T : \mathcal{L}_0 \to \mathcal{L}_0$ such that for any $\varphi \in \mathcal{L}_0$,

- $T(\varphi)$ is in CNF.
- φ is satisfiable iff $T(\varphi)$ is satisfiable.
- T runs in time $O(len(\varphi))$, i.e., linear in the size of φ.

The standard algorithm for this is called *Tseitin Transformation*.

Remark 2.1.1 (Tseitin Transformation)**.** The basic idea is that you can always introduce new Boolean variable, say b, to represent a subformula ψ, and add a constraint $b \leftrightarrow \psi$ to enforce their equivalence. The beautiful part is that the equivalence condition is a "high-level CNF":

$$b \leftrightarrow \psi \;\equiv\; (b \to \psi) \wedge (\psi \to b) \;\equiv\; (\neg b \vee \psi) \wedge (\neg \psi \vee b).$$

Thus, by doing this operation recursively and smooth out the inside of ψ, we get a CNF formula with a bunch of new variables. The full algorithm is in Algorithm 1.

Remark 2.1.2. So from now on, without loss of generality, we only need to design SAT-solving algorithms for solving CNF formulas.

Notation 2.1.1 (Clauses)**.** Since we restrict the input formulas to be in CNF, we can simplify notations. From now on, we write formulas as a set of *clauses*, where each clause is a set of literals. Namely, for any φ in the form of Definition 2.1.1, we write

$$\varphi := \{..., \omega_i, ...\} \text{ and } \omega_i =: \{l_{i1}, ..., l_{ik_i}\}.$$

Thus it makes sense to write $\omega \in \varphi$, $l \in \omega$, etc. We also write $var(\varphi)$ to denote the set of all propositional variables in φ.

Algorithm 1 Tseitin Transformation

Using De Morgan laws, we can also put any formula in the form of "disjunctions of conjunctions" which is called the Disjunctive Normal Form (DNF).

Definition 2.1.2 (Disjunctive Normal Form)**.** We say $\varphi \in \mathcal{L}_0$ is in Disjunctive Normal Form (DNF), if it is of the form:

$$\varphi := \bigvee_i (l_{i1} \wedge \cdots \wedge l_{ik_i})$$

where $l_{ij} = p_{ij}$ or $l_{ij} = \neg p_{ij}$ for some $p_{ij} \in P(\mathcal{L}_0)$.

Remark 2.1.3. Satisfiability can be determined in *linear time* for DNF. The catch is that, assuming $\mathsf{P} \neq \mathsf{NP}$, there does not exist any polynomial-time algorithm that turns any formula into an equisatisfiable DNF formula. On the other hand, checking validity of DNF formulas is NP-complete while checking validity of CNF formulas is in linear time.

Exercise 2.1.1. Prove that any formula ϕ is equivalent to a DNF formula whose size is $O(N_\varphi)$, where N_φ is th number of satisfiying assignments of ϕ.

2.2 Davis-Putnam-Logemann-Loveland (DPLL)

The starting point of modern SAT solving algorithms was introduced in 1962, named after its inventors. The key contribution is that formulation of the unit propagation rule. Its backtracking scheme is very naive, compared to later development in CDCL algorithms.

Unit propagation comes from the following observation. Consider any clause

$$\omega = \{l_1, ..., l_k\}.$$

Suppose, out of the k literals, $k-1$ literals have been assigned 0, then the last literal has to be assigned 1. This is sometimes referred to as the *unit rule* and it is intuitively straightforward: the only way to ensure that ω evaluates to 1 is to have at least one literal evaluated to 1.

Definition 2.2.1 (Unit Clause)**.** Let $\omega = \{l_1, ..., l_k\}$ be a clause and σ be an assignment. If $\sigma(l_1) = \cdots = \sigma(l_{k-1}) = 0$, then we say ω is a *unit clause*.

Algorithm 2 DPLL Search

```
1: function DPLL(φ)
2:     σ ← ∅                                          ▷ Assignment mapping
3:     Δ ← ∅                                          ▷ Decision variable stack
4:     φ ← PureLiteralElimination(φ)
5:     while true do
6:         σ ←UnitPropagate(ϕ, σ)
7:         if ⊥ ∉ σ then
8:             if AllAssigned(σ) then
9:                 return Satisfiable
10:            else
11:                ⟨b, p⟩ ← MakeDecision(ϕ, σ)
12:                Δ ← Δ ∪ ⟨b, p⟩
13:                σ = σ ∪ ⟨b, p⟩
14:            end if
15:        else
16:            if Δ == ∅ then
17:                return unsatisfiable
18:            else
19:                ⟨b, p⟩ ← Δ.pop_back()
20:                σ ← σ ∪ ⟨b, ¬p⟩
21:            end if
22:        end if
23:    end while
24: end function
```

Remark 2.2.1. Unit propagation, which is sometimes referred to as Boolean Constraint Propagation (BCP), is the main "driving engine" in DPLL-style SAT solving procedure. Note that it only works when the problem is in CNF.

The authors of DPLL defined other simple rule as well.

Definition 2.2.2 (Pure Literals)**.** We say l is a pure literal in a formula φ if $\neg l$ does not appear in φ.

Proposition 2.2.1. Any pure literal can be directly assigned to 1.

The DPLL algorithm is then a straightforward depth-first search that uses unit propagation. First, preprocess the formula to eliminate all pure literals and the clauses that contain them. Next, start the following loop: if there are unit clauses, then update the assignment map with the corresponding values based on the unit rule; if there are no more unit clauses while some variables are still unassigned, make a decision on one of the free variables. Within the loop, when all variables have been assigned values with no conflict, return "satisfiable." On the other hand, if a decision leads to a conflict, then immediately backtrack and revert the decision. If no other decisions can be made (backtracked to the top level), then return unsatisfiable.

Algorithm 3 Unit Propagation

```
function UnitPropagate(φ, σ)
    while true do
        ω ← PickActiveClause(φ, σ)
        if ω!= NULL then
            ⟨p, b⟩ ← UnitRule(ω, σ)
            σ ← σ ∪ ⟨p, b⟩
        else
            return σ
        end if
        σ ← ∅
        dl ← 0
    end while
end function
```

Watched Literals. We need an important technique for pick active clauses.

Definition 2.2.3 (Watched Literals)**.**

Remark 2.2.2. No need to rewrite anything when backtracking.

2.3 Conflict-Driven Clause-Learning Search (CDCL)

At a high-level, CDCL search is described as in Algorithm 5, followed by the definitions.

Definition 2.3.1 (Assignment Set)**.** An assignment set σ is a set of pairs $\langle p, b\rangle$ where $p \in P$ and $v \in \{0, 1\}$. Each p appears uniquely in σ: for any $\langle p_1, b_1\rangle, \langle p_2, b_2\rangle \in \sigma$, $p_1 \neq p_2$.

Definition 2.3.2 (Implication Graph)**.** An implication graph is a directed labeled graph $IG = \langle V, E, l\rangle$ of the following form:

- The vertex set V consists of tuples $\langle p, b, d\rangle$ where $p \in P(\mathcal{L}_0)$ is a propositional variable, $b \in \{0, 1\}$, and $d \in \mathbb{N}$ is called the *decision level*.
- V can also contain a special symbol $\perp$ that indicates logical contradiction.
- The edges $\langle v_1, v_2\rangle \in E$ are directional, indicating logical implications.
- The labeling function maps each edge $e \in E$ to some clause $\omega \in \varphi$.

When V contains $\perp$, for notational simplicity we also write $\perp \in IG$.

Definition 2.3.3 (AddImplication)**.** Add a vertex $\langle p, b, dl\rangle$ in IG, and create edges from all vertices that contain the other variables in ω to the newly added vertex. Label these new edges with ω.

Algorithm 4 CDCL Search

```
1: function CDCL(φ)
2:     σ ← ∅
3:     dl ← 0
4:     IG ← ∅
5:     ⟨σ, IG⟩ ← UnitPropagation(φ, σ, IG, dl)
6:     if ⊥ ∈ IG then
7:         return unsat
8:     end if
9:     while not AllAssigned(φ, σ) do
10:        ⟨p, b⟩ ← PickBranchingVariable(φ, σ, IG)
11:        σ ← σ ∪ {⟨p, b⟩}
12:        dl ← dl + 1
13:        IG ← ⟨V ∪ {⟨p, b, dl⟩}, E, l⟩
14:        ⟨σ, IG⟩ ← UnitPropagation(φ, σ, IG, dl)
15:        if ⊥ ∈ IG then
16:            dl ← ConflictAnalysis(φ, σ, IG)
17:            if dl < 0 then
18:                return unsat
19:            else
20:                ⟨σ, IG⟩ ← Backtrack(φ, σ, dl, IG)
21:            end if
22:        end if
23:    end while
24:    return sat
25: end function
```

Example 2.3.1. Generate an implication graph.

Definition 2.3.4 (AllAssigned)**.** Return true if all variables in φ appear in σ.

Definition 2.3.5 (PickBranchingVariable)**.** Return a propositional variable in $var(\varphi)$ but not yet included in σ, and give it a value $b \in \{0, 1\}$. Requires heuristics.

Conflict Analysis. It is the other most important subroutine in the algorithm. More details next time.

Definition 2.3.6 (Backtrack)**.** Clean up the implication graph based on conflict analysis.

Algorithm 5 Unit Propagation in CDCL

```
1: function UnitPropagationCDCL(φ, σ, IG, dl)
2:     while true do
3:         ω ← PickActiveClause(φ, σ)
4:         if ω != NULL then
5:             ⟨p, b⟩ ← UnitRule(ω, σ)
6:             σ ← σ ∪ ⟨p, b⟩
7:             IG ← IG ∪ AddImplication(ω, ⟨p, b⟩, dl)
8:         else
9:             return ⟨σ, IG⟩
10:        end if
11:        σ ← ∅
12:        dl ← 0
13:    end while
14: end function
```

Part II

First-Order Reasoning

Chapter 3

First-Order Logic and Theories

3.1 Syntax

Notation 3.1.1 (Signature)**.** The signature of a first-order language is a pair of sets $\langle F, R \rangle$. Here, F represents the set of all *function* symbols:

$$F := \{f_j^i \mid i, j \in \mathbb{N}\}$$

where the superscript is the *arity* of a function. Similarly, R represents the set of all *predicate* symbols:

$$R := \{P_j^i \mid i, j \in \mathbb{N}\}.$$

Remark 3.1.1. Constants are 0-ary functions. Truth and falsity are 0-ary predicates.

Definition 3.1.1 (Terms)**.** Let X be the set of *variables*:

$$X := \{x_i \mid i \in \mathbb{N}\}.$$

We define the set of terms T to consist of elements of the form:

$$t := x \mid f^i(t_1, ..., t_i)$$

for arbitrary $x \in X$ and $f^i \in F$.

Definition 3.1.2 (Formulas)**.** We define $\mathcal{L}_1$ to be the set of all formulas of the form:

$$\varphi := P^i(t_1, ..., t_i) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x\varphi$$

for arbitrary $P^i \in R$.

Here, $\exists$ is an *existential* quantifier. We can use it to define the *universal* quantifier.

Definition 3.1.3 (Syntactic Sugar: Universal Quantifier)**.** We define

$$\forall x\varphi := \neg\exists x\neg\varphi.$$

3.2 Semantics

Definition 3.2.1 (Structures)**.** A structure is a tuple $\mathcal{M} = \langle D, F^{\mathcal{M}}, R^{\mathcal{M}} \rangle$ where

- D is the domain of interpretation for the language.
- $F^{\mathcal{M}}$ is the set of functions over D that provides interpretations for all function symbols in F, namely:
$$F^{\mathcal{M}} := \{(f^i)^{\mathcal{M}} : D^i \to D \mid f^i \in F\}.$$
- Similarly, $R^{\mathcal{M}}$ is the set of predicates that interpret all symbols in R:
$$R^{\mathcal{M}} := \{(P^i)^{\mathcal{M}} \subseteq D^i \mid P^i \in R\}.$$

Definition 3.2.2 (Models)**.** Let $\mathcal{M}$ be a structure. A model is a pair
$$M := \langle \mathcal{M}, \sigma \rangle$$
where
$$\sigma : X \to D$$
is an interpretation function from the set of variable symbols to the domain of interpretation.

Definition 3.2.3 (M-Interpretation of Terms)**.** Let M be a model. The M-interpretation function of all terms in $\mathcal{L}_1$ (write the set of all terms as $T(\mathcal{L}_1)$)
$$[\![\cdot]\!]^M : T(\mathcal{L}_1) \to D$$
is inductively defined as:

- $[\![x]\!]^M := \sigma(x)$.
- $[\![f(t_1, ..., t_n)]\!]^M := f^M([\![t_1]\!]^M, ..., [\![t_n]\!]^M)$.

Definition 3.2.4 (M-Interpretation of Formulas)**.** Let M be a model and $\mathcal{M}$ be its structure. The M-interpretation function of formulas in $\mathcal{L}_1$
$$[\![\cdot]\!]^M : \mathcal{L}_1 \to \{0, 1\}$$
is inductively defined as:

- $[\![P(t_1, ..., t_n)]\!]^M := 1$ iff $([\![t_1]\!]^M, ..., [\![t_n]\!]^M) \in P^{\mathcal{M}}$.
- $[\![\neg\varphi]\!]^M := 1 - [\![\varphi]\!]^M$.
- $[\![\varphi_1 \wedge \varphi_2]\!]^M := [\![\varphi_1]\!]^M \cdot [\![\varphi_2]\!]^M$.
- $[\![\exists x\varphi]\!]^M := 1$ iff there exists some $a \in D$, such that when we force the interpretation of x to be a in M (written as $M[x \mapsto a]$), we have $[\![\varphi]\!]^{M[x \mapsto a]} = 1$.

Definition 3.2.5 (Satisfiability and Validity)**.** A first-order logic formula φ is satisfiable in structure $\mathcal{M}$ if there exists an $\mathcal{M}$-model M such that $[\![\varphi]\!]^M = 1$. We say φ is valid if for any $\mathcal{M}$-model M, $[\![\varphi]\!]^M = 1$.

Example 3.2.1. First-order languages over integers and over real numbers.

Chapter 4

Constraint Satisfaction

4.1 Constraint Propagation

4.2 Monte Carlo Methods

Chapter 5

Mathematical Optimization

5.1 Convex Optimization

5.1.1 Interior Point Methods

5.2 Nonlinear Optimization

5.2.1 Gradient Descent

5.3 Mixed-Integer Optimization

Chapter 6

Delta-Decision Procedures

6.1 Nonlinear Theories

6.2 Delta-Completeness

6.3 Quantified Reasoning

Part III

Probabilistic Reasoning