Notes for Verification and Validation Techniques for Artificial Intelligence and Cybersecurity

Stefano Trevisani

July 25, 2022

Contents

1 Introduction to Logic

Logic is a formal language to reason about the properties of some 'objects'. A logic, as any formal language, must have:

- *Lexicon* (or *vocabulary*): symbols to denote objects and operations.
- *Syntax*: rules to combine symbols into constructs (terms, formulae).
- *Semantic*: meaning assigned to the syntactic constructs.
- *Algorithms* (or *operations*): transformations for reasoning about the semantic of a syntactic construct.

A *tautology* (or *valid formula*) is a formula which is true in every model.

Example 1. *Consider the following proposition:*

> *All men are mortal. Socrates is a man. Therefore, Socrates is mortal.*

It is a specific case of the more generic tautology:

$$(\forall x \colon A(x) \Rightarrow B(x)) \land A(y) \Rightarrow B(y)$$

A *contradiction* (or *paradox*) is a formula which is false in every model.

Example 2. *Consider the following proposition:*

> *There is a barber which shaves everyone who doesen't shave themselves.*

It is a specific case of the more generic paradox:

$$\exists x, \forall y \colon A(x, y) \Leftrightarrow \neg A(y, y)$$

The variables in a formula can be of different *sort*, as they represent objects of different *type*.

Example 3. *Consider the following proposition:*

> *Every incoming order is eventually processed*

It can be encoded in the following formula:

$$\forall o \in O, \forall t \in T \colon arrives(o, t) \Rightarrow \exists t' \in T \colon t' > t \land process(o, t')$$

1.1 Problems on logic

There are several problems which can be defined over a logic:

- *Evaluation* (or *Model checking*, EVAL): given a formula ϕ and an interpretation M, decide whether $M \models \phi$ (i.e. whether M is a model of ϕ).
- *Satisfiability* (SAT): given a formula ϕ, decide whether $\exists M \colon M \models \phi$.
- *Validity* (VAL): given a formula ϕ, decide whether $\forall M \colon M \models \phi$.
- *Equivalence* (EQUIV): given two formulae ϕ, ψ, decide whether $\phi \equiv \psi$.
- *Consequence* (CONS): given two formulae ϕ, ψ, decide whether $\phi \Longrightarrow \psi$.

Different logics can have similar problems associated with them, but they often differ in *expressive power* and *succintness*: problems associated with more expressive logics belong to higher complexity classes, and can even be undecidable, while succint logics offer more compact representations of similar problems, (artificially) increasing the associated complexity.

1.2 Propositional logic

Propositional logic is the simplest kind of formal logic:

- Its *lexicon* consists of propositional symbols, tipically denoted with lowercase letters $p, q, r, \ldots$ and the boolean connectives $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$. The set of all propositional symbols (the *alphabet*) is denoted with Σ.
- Its *syntax* consists of formulae denoted with greek letters $\phi, \psi, \ldots$ such that:
$$\phi := p \mid \neg\psi \mid \psi \wedge \psi' \mid \psi \vee \psi' \mid \psi \Rightarrow \psi' \mid \psi \Leftrightarrow \psi'$$
- Its *semantic* consists of an *interpretation* function $M \colon \Sigma \mapsto \{\bot, \top\}$, which *holds* on (or *models*, or *entails*) a formula ϕ:

$$\begin{array}{lcl}
M \models p & \Longleftrightarrow & M(p) = \top \\
M \models \neg\phi & \Longleftrightarrow & M \not\models \phi \\
M \models \phi \wedge \psi & \Longleftrightarrow & M \models \phi \mathbin{\&} M \models \psi \\
M \models \phi \vee \psi & \Longleftrightarrow & M \models \phi \mathbin{\curlyvee} M \models \psi \\
M \models \phi \Rightarrow \psi & \Longleftrightarrow & M \models \phi \Longrightarrow M \models \psi \\
M \models \phi \Leftrightarrow \psi & \Longleftrightarrow & M \models \phi \Longleftrightarrow M \models \psi
\end{array}$$

Furthermore:

$$\begin{array}{lcl}
\phi \text{ is a } \textit{tautology} & \Longleftrightarrow & \forall M \colon M \models \phi \\
\phi \text{ is a } \textit{contradiction} & \Longleftrightarrow & \forall M \colon M \not\models \phi \\
\phi \text{ is a } \textit{consequence} \text{ of } \psi & \Longleftrightarrow & \forall M \colon M \models \psi \Longrightarrow M \models \phi \\
\phi \text{ is } \textit{equivalent} \text{ to } \psi & \Longleftrightarrow & \forall M \colon M \models \phi \Longleftrightarrow M \models \psi
\end{array}$$

Remark 1. Given two formulae ϕ and ψ and the proposition $\phi \Rightarrow \psi$:

- $\phi \Rightarrow \psi$ is the *direct* proposition.
- $\psi \Rightarrow \phi$ is the *converse* proposition.
- $\neg\phi \Rightarrow \neg\psi$ is the *inverse* proposition.
- $\neg\psi \Rightarrow \neg\phi$ is the *contrapositive* proposition.
- If the direct is false, the inverse is true.
- If the direct is true, the contrapositive is true.

To solve the model checking problem for propositional logic, we can use Algorithm 1, which is a textbook recursive divide-and-conquer algorithm taking $\mathcal{O}(|\phi|)$ time (assuming that evaluating $M(p)$ takes constant time).

Algorithm 1 An algorithm to evaluate a propositional formula

Require: A NNF formula ϕ and an interpretation M
Ensure: Whether M is a model for ϕ

```
function modelCheck(φ, M)
    switch φ
        case p
            if M(p) = ⊤
                return true
            else
                return false
        case ¬φ1
            return ¬modelCheck(φ1, M)
        case φ1 ∧ φ2
            return modelCheck(φ1, M) ∧ modelCheck(φ2, M)
        case φ1 ∨ φ2
            return modelCheck(φ1, M) ∨ modelCheck(φ2, M)
```

It is well known that EVAL is a **P**-complete problem[1], and that it is possible to convert any propositional formula to an equivalent NNF formula in polynomial time.

On the other hand, to solve the SAT problem for propositional logic, we can use Algorithm 2, which takes exponential time to execute.
It is well known that SAT for propositional logic is an **NP**-complete problem[2].

[1] $\mathbf{P} = TIME(\mathcal{O}(N^k))$ is the class of problems decidable by a deterministic Turing machine in time at most polynomial to the input size N.

[2] $\mathbf{NP} = NTIME(\mathcal{O}(N^k))$ is the class of problems decidable by a non-deterministic Turing machine in time at most polynomial to the input size N.

Algorithm 2 An algorithm to decide satisfiability of a propositional formula

Require: A NNF formula ϕ
Ensure: Whether ϕ is satisfiable or not
  function isSatisfiable(ϕ)
    $M \leftarrow (\bot)^{|\phi|}$
    while $M \neq (\top)^{|\phi|}$
      if modelCheck(ϕ, M) = **true**
        return true
      $M \leftarrow$ nextCombination(M)
    if modelCheck(ϕ, M) = **true**
      return true
    return false

1.2.1 Normal forms

All propositional formulae can be rewritten in one of the so-called *normal forms*, which reduce the vocabulary and constrain the syntactic structure of formulae with the advantage of making them easier to work on. There are three main normal forms:

- *Negation Normal Form* (NNF): the only allowed connectors are $\neg$, $\wedge$ and $\vee$. Since $x \Rightarrow y \equiv \neg x \vee y$ and $x \Leftrightarrow y \equiv x \Rightarrow y \wedge y \Rightarrow x$, we do not lose expressive power. There are no restrictions on the syntactic structure of a formula. If a formula ϕ contains two-way implications, then $|NNF(\phi)| = \mathcal{O}\big(2^{|\phi|}\big)$, otherwise it will always be $\mathcal{O}(|\phi|)$.

- *Conjunctive Normal Form* (CNF): the allowed connectors are the same of NNF, but all the conjuctions must be at the top level of a formula:

$$\phi = (l_{1,1} \vee \cdots \vee l_{1,n_1}) \wedge \cdots \wedge (l_{m,1} \vee \cdots \vee l_{m,n_m})$$

  where $l = p \mid \neg p$. It is always possible to translate any formula in an equivalent CNF formula using De Morgan's laws and distributive laws:

  - $x \vee y \equiv \neg(\neg x \wedge \neg y)$
  - $x \wedge y \equiv \neg(\neg x \vee \neg y)$
  - $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
  - $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$

  Unfortunately, $|CNF(\phi)| = \mathcal{O}\big(2^{|\phi|}\big)$ in general, but it is possible to introduce new additional variables to keep the size linear: we lose *equivalence* but we keep *equisatisfiability* (CNF-SAT is still **NP**-complete). This form is also known as *Product of Sums* (POS) form.

- *Disjunctive Normal Form* (DNF): the allowed connectors are the same of NNF, but all the disjunctions must be at the top level of a formula:

$$\phi = (l_{1,1} \wedge \cdots \wedge l_{1,n_1}) \vee \cdots \vee (l_{m,1} \wedge \cdots \wedge l_{m,n_m})$$

where $l = p \mid \neg p$. It is always possible to translate any formula in an equivalent DNF formula using De Morgan's laws and distributive laws. In general, $|DNF(\phi)| = \mathcal{O}\big(2^{|\phi|}\big)$, and there is no known satisfiability-preserving polynomial space transformation (in fact, DNF-SAT is in $\mathbf{P}$). This form is also known as *Sum of Products* (SOP) form.

1.2.2 Tableaux method

To solve the satisfiability problem for propositional logic, we saw that we could use Algorithm 2, assuming the formula was in NNF. A better approach to the SAT problem is the *tableaux method*, which is a special case of the backtracking method. More generally, backtracking algorithms are devised to solve *Constraint Satisfaction Problems* (CSPs), i.e. search problems involving a set variables $V = \{x_1, \ldots, x_n\}$ each over some domain $\mathcal{D} = \{D_1, \ldots, D_n\}$ subject to a set of constraints C. For propositional logic, $D_1 = \cdots = D_n = \{\bot, \top\}$, and the constraints are determined by the logical connectors between the subformulae. If we a formula ϕ left-to-right, we can see the search space as a binary tree where at each level k we split the tree in two branches if $\phi_k = \phi_{k+1,l} \vee \phi_{k+1,r}$ (we assume the left side to be true in the left branch, and the right side to be true in the right branch) or we split the formula if $\phi_k = \phi_{k+1,l} \wedge \phi_{k+1,r}$ (as both sides must be true). Every leaf of the tree will then represent an interpretation, which becomes a model if the formula is satisfied. While the tree can contain up to 2^n leaves, this does not always happen, as partial assignments might cause an early invalidation of the formula. The asymptotical complexity is still exponential, of course, but backtracking and tableaux methods can exploit many kinds of heuristics which often drastically reduce the search time.

1.2.3 Transition systems

Consider a k-bit finite state machine $\mathcal{M}$, which can be in one of $n = 2^k$ possible states $S_i = \big(\lceil \frac{i}{2^{k-1}} \rceil \bmod 2, \ldots, \lceil \frac{i}{2^0} \rceil \bmod 2\big)$. We can describe the *reachability* problem REACH as follows:

- A formula $\phi_{init}(X)$ which describes whether a state can be the initial state of the system.
- A formula $\phi_{goal}(X)$ which describes whether a state can be the final target of the system.
- A formula $\phi_{tran}(X, Y)$ which describes if it is possible for the system to transition from the first state to the second state.
- A formula $\phi_{same}(X, Y) = \bigwedge_{i=0}^{k}(X_i \Leftrightarrow Y_i)$, which describes whether two states are the same. It is not needed if $\phi_{trans}(X, X)$ is always true.

Then, the formula which describes the reachability problem from an initial state X_1 to a final state X_n passing by states $X_2, \ldots, X_{n-1}$ is simply:

$$\phi(X_1, \ldots, X_n) = \phi_{init}(X_1) \wedge \phi_{goal}(X_n) \wedge \bigwedge_{i=1}^{n} \phi_{tran}(X_i, X_{i+1}) \vee \phi_{stay}(X_i, X_{i+1})$$

Note that, in ϕ, we didn't just fix the initial and final states, but also all the intermediate states: if there is an alternative path to reach the final state, it will not be taken into account from our formula. Furthermore, if $\mathcal{M}$ cannot get from a state A to a state B in less than 2^k steps, then there is no way to get from A to B.

Example 4. *Suppose we have a 4-bit finite state machine $\mathcal{M}$, then we have $2^4 = 16$ different possible states. We define $\phi_{init}(X)$ as:*

$$\phi_{init}(X) = \neg X_2 \wedge \neg X_3 \wedge \neg X_4$$

The only states which satisfy ϕ_{init} are $S_0 = (0,0,0,0)$ and $S_1 = (1,0,0,0)$. Then we define ϕ_{goal} as:

$$\phi_{goal}(X) = X_1 \wedge X_2 \wedge X_3 \wedge X_4$$

Which is satisfied only by $S_{15} = (1,1,1,1)$[3].

Suppose that our machine is a counter, then the transition formula will be:

$$\begin{aligned}
&\phi_{tran}(X, Y) = \\
&(\neg X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4 \vee \neg Y_1) \wedge (X_1 \vee \neg X_2 \vee \neg X_3 \vee \neg X_4 \vee Y_1) && \wedge \\
&(\neg X_1 \vee Y_1 \vee \neg Y_2) \wedge (X_1 \vee \neg Y_1 \vee \neg Y_2) \wedge (\neg X_1 \vee X_2 \vee Y_1) \wedge (X_1 \vee X_2 \vee \neg Y_1) && \wedge \\
&(\neg X_2 \vee Y_2 \vee \neg Y_3) \wedge (X_2 \vee \neg Y_2 \vee \neg Y_3) \wedge (\neg X_2 \vee Y_2 \vee \neg Y_4) \wedge (X_2 \vee \neg Y_2 \vee \neg Y_4) && \wedge \\
&(\neg X_3 \vee Y_3 \vee \neg Y_4) \wedge (X_3 \vee \neg Y_3 \vee \neg Y_4) \wedge (X_3 \vee \neg X_4 \vee Y_3) && \wedge \\
&(X_2 \vee \neg X_3 \vee \neg X_4 \vee Y_2) \wedge (X_4 \vee Y_4)
\end{aligned}$$

This formula is satisfied only when applied to S_i and $S_{i+1 \bmod 16}$, as it states that the 4-bit number Y is the successor (modulo 16) of the 4-bit numer X. Now, let's check some cases when trying to satisfy $\phi(X_1, \ldots, X_n)$:

- *If we choose $X_i = S_i$, then ϕ is satisfied, as S_0 is a valid initial state (ϕ_{init} is satisfied), S_n is a valid final state (ϕ_{goal} is satisfied), and it is possible to transition from S_i to S_{i+1}, (all occurences of ϕ_{tran} are satisfied).*
- *If $X_{15} \neq S_{15}$, ϕ_{goal} won't be satisfied, as S_{15} is the only valid final state.*
- *Similarly, ϕ_{init} will never be satisfied if X_1 is neither S_1 nor S_8.*
- *Again, some ϕ_{tran} won't be satisfied if, when $X_i = S_j$, $X_{i+1} \neq S_{j+1}$. However, if $X_{i+1} = X_i$, then ϕ_{stay} is still satisfied.*
- *Assume that $X_1 = S_8$ and $X_{15} = S_{15}$, it is possible to transition from S_8 to S_{15} in $7 < 15$ steps. Then, we set $X_2 = S_9, \ldots, X_7 = S_{14}$ and $X_8 = \cdots = X_{15}$: thanks to ϕ_{stay}, the whole formula is satisfied.*

[3]Since $\{\bot, \top\}$ is isomorphic to $\{0, 1\}$, we use the two sets interchangeably.

1.3 Quantified propositional logic