CS 337, Fall 2023

Logic - II

Scribes: Abhijit Amrendra Kumar, Bitra Sai Siddhi Raghuram Saran, Duggineni Venkata Paneesh, Rayane Tayache, Tanay Tayal, Pratiksha Deka, Sathwika Reddy, Desai Sai Pranav*
Edited by: Vishal Tapase

November 2, 2023

Disclaimer. Please note this document has not received the usual scrutiny that formal publications enjoy. This may be distributed outside this class only with the permission of the instructor.

1 Logic (continued)

1.1 Deductive implications

Given an entailment $P \rightarrow Q$ there are 3 other important implications that follow

- Converse : $Q \rightarrow P$
- Inverse : $\neg P \rightarrow \neg Q$
- Contraposition: $\neg Q \rightarrow \neg P$

1.2 Types of sentences

- A valid sentence / Tautology / $\top$ is true in all models
- An inconsistent sentence / Contradiction / $\perp$ is false in all models
- A satisifiable sentence is true in some models (non-zero)

$$\text{eg : } \alpha \rightarrow \beta \quad \text{iff} \quad \alpha \wedge \neg\beta \quad \text{is unsatisifiable}$$

(proof by contradiction)

Implication: $\alpha \rightarrow \beta$ if and only if $\alpha \wedge \neg\beta$ is unsatisfiable.

Proof:

1. First, let's show that if $\alpha \rightarrow \beta$, then $\alpha \wedge \neg\beta$ is unsatisfiable. Assume $\alpha \rightarrow \beta$. We want to show that $\alpha \wedge \neg\beta$ is unsatisfiable, which means there is no assignment of truth values to the variables in α and β that makes $\alpha \wedge \neg\beta$ true.

   Suppose, for the sake of contradiction, that there is an assignment that satisfies $\alpha \wedge \neg\beta$. This would mean that α is true and $\neg\beta$ is true under the same assignment. But if α is true, then by the definition of implication, $\alpha \rightarrow \beta$ must also be true, which means β is true. However,

*All scribes get equal credit for the final notes.

this contradicts the fact that $\neg\beta$ is also true in the same assignment. Therefore, we have a contradiction, and $\alpha \wedge \neg\beta$ cannot be satisfied, which means it is unsatisfiable.

2. Next, let's show that if $\alpha \wedge \neg\beta$ is unsatisfiable, then $\alpha \rightarrow \beta$:

   Assume $\alpha \wedge \neg\beta$ is unsatisfiable, which means there is no assignment of truth values to the variables in α and β that makes $\alpha \wedge \neg\beta$ true. This implies that there is no assignment where both α is true and $\neg\beta$ is true simultaneously.

   Now, let's prove by contradiction that $\alpha \rightarrow \beta$ is true. Suppose $\alpha \rightarrow \beta$ is false, which means there is an assignment where α is true, but β is false. This assignment would also satisfy $\alpha \wedge \neg\beta$, which contradicts our initial assumption that $\alpha \wedge \neg\beta$ is unsatisfiable.

Thus, we have established both directions of the equivalence:

1. If $\alpha \rightarrow \beta$, then $\alpha \wedge \neg\beta$ is unsatisfiable.
2. If $\alpha \wedge \neg\beta$ is unsatisfiable, then $\alpha \rightarrow \beta$.

Hence, we have proven that $\alpha \rightarrow \beta$ if and only if $\alpha \wedge \neg\beta$ is unsatisfiable.

1.3 Types of inference rules

- A sound inference rule derives only entailed sentences
- A complete inference rule can derive any entailed sentences
- We want inference rules to be sound & complete. A rule with such properties is called RESOLUTION

Note: Modus Ponens is sound but not complete. Here is an example.

Consider the following KB=$\{W \rightarrow J, J \rightarrow B, B \rightarrow O\}$

Our goal is to determine whether O follows from W, without directly using the premise W. We initially apply Modus Ponens as follows:

1. From premise 1 ($W \rightarrow J$) and the fact that W is true, we can apply Modus Ponens to derive J:

$$(W \rightarrow J), W \models J$$

2. From premise 2 ($J \rightarrow B$) and the fact that J is true (derived in the previous step), we can apply Modus Ponens to derive B:

$$(J \rightarrow B), J \models B$$

However, it's important to note that Modus Ponens alone does not allow us to directly conclude O from W. We cannot use Modus Ponens again to establish O because there is no direct premise linking B to O without introducing a new premise.

To bridge the gap between B and O when W, we need to apply the transitive property of implication, which combines the premises. The transitive property can be expressed as follows:

$$\text{If } A \to B \text{ and } B \to C, \text{ then } A \to C$$

In our case, we have:

$$(W \to J) \text{ and } (J \to B), \text{ and we want to conclude } (W \to B).$$

Using the transitive property of implication, we can combine the premises:

$$(W \to J) \text{ and } (J \to B) \models (W \to B)$$

So, by applying the transitive property of implication, we can conclude $(W \to B)$ when W, which is equivalent to saying that B follows from W without directly using the premise W.

While Modus Ponens is a fundamental and valid inference rule, it is not complete for proving the conclusion O when starting with W. Additional reasoning in the form of the transitive property of implication is needed to establish the link between B and O. Therefore, Modus Ponens alone is not sufficient to demonstrate the completeness of the proof in this context.

2 Resolution

Resolution is an inference rule that is both sound and complete. A resolution produces a new clause (i.e. disjunction of literals) implied by two clauses that contain complementary literals.

$$\frac{(\alpha \lor \beta), (\neg\beta \lor \gamma)}{\alpha \lor \gamma}$$

To apply resolution to a KB, it needs to be in Conjunctive Normal Form(**CNF**). A sentence is said to be in CNF if it can be expressed as a conjunction of clauses.

Here's an example of converting a sentence to CNF.

$$\begin{aligned}
(\alpha \lor \beta) \leftrightarrow \gamma \quad &\equiv \quad (\alpha \lor \beta) \to \gamma \quad \land \quad \gamma \to (\alpha \lor \beta) \\
&\equiv \quad (\neg(\alpha \lor \beta) \lor \gamma) \quad \land \quad (\neg\gamma \lor \alpha \lor \beta) \\
&\equiv \quad ((\neg\alpha \land \neg\beta) \lor \gamma) \quad \land \quad (\neg\gamma \lor \alpha \lor \beta) \\
&\equiv \quad (\neg\alpha \lor \gamma) \quad \land \quad (\neg\beta \lor \gamma) \quad \land \quad (\neg\gamma \lor \alpha \lor \beta)
\end{aligned}$$

If we want to show that $KB \models \alpha$, then we have to show $(KB \land \neg\alpha)$ is unsatisfiable

1. Write $(KB \land \neg\alpha)$ in CNF
2. Apply the resolution rule to pairs of clauses with complementary literals to produce a new clause
3. Continue this until either of these happens:
   (a) no new clauses are added.$(KB \not\models \alpha)$
   (b) an empty clause(a contradiction) is derived.

- Resolution is both **sound** and **complete** for KB's (in CNF).

- An efficient version of resolution can be implemented using **forward-chaining** if the clauses in the KB are **definitive** clauses.
- A **definite** clause is a Horn clause with exactly one positive literal (**Horn clause** refers to a disjunction of literals in which, at most, one literal is not negated.). In other words, it is a disjunction of literals with exactly one literal being positive. ($e.g : \neg\alpha \vee \neg\beta \vee \gamma$).
- A definitive clause can be written as an implication, where the premise is a conjuction of positive literals and conclusion is a positive literal. ($e.g : \alpha \wedge \beta \to \gamma$)

Algorithm to convert a formula to CNF:

Algorithm 1 Conversion to Conjunctive Normal Form (CNF)

---

1. **Push Negations**: Push negations into the formula, repeatedly applying De Morgan's Law, until all negations only apply to atoms (Negative Normal Form).

    while negations are present and not applied to atoms **do**
        Apply De Morgan's Law to negate disjunctions and conjunctions
        Apply double negation elimination
    end while

    Example: $\neg(p \vee q)$ becomes $(\neg p) \wedge (\neg q)$

2. **Apply Distributive Law**: Repeatedly apply the distributive law where a disjunction occurs over a conjunction. Continue until this is no longer possible, and the formula is in CNF.

    while disjunction over conjunction is present **do**
        Apply distributive law
    end while

    Example: $p \vee (q \wedge r)$ becomes $(p \vee q) \wedge (p \vee r)$

To show that Knowledge Base entails a formula α i.e $KB \models \alpha$, we need to show that $(KB \cap \neg\alpha)$ is unsatisfiable
Algorithm for finding if $KB \models \alpha$:

Algorithm 2 Checking if $KB \models \alpha$

---

procedure CheckEntailment(KB, α)
    Convert KB and $\neg\alpha$ into CNF
    while true **do**
        Apply resolution rule to pairs of clauses with complementary symbols (β and $\neg\beta$) to produce a new clause
        if No new clauses are added **then** // *Saturated*
            return KB doesn't entail α
        else if An empty clause (contradiction) is derived **then**
            return KB entails α
        end if
    end while
end procedure

Note: Resolution is guaranteed to terminate as closure of resolution is finite.

2.1 Completeness Proof

Completeness is proven using semantic trees for an unsatisfiable clause set N. We will use structural induction on the size of the semantic tree.

Semantic Tree

A semantic tree, also known as a truth tree or tableau tree, is a graphical representation used in mathematical logic. Semantic trees are primarily employed for assessing the truth values and logical consequences of logical formulas and statements. They are commonly used in both propositional logic and first-order logic.
Let's illustrate how to construct a semantic tree for a propositional logic formula.
Consider the following formula: $(P \land Q) \lor (\neg R \land S)$.

$$
\begin{array}{cccc}
\multicolumn{4}{c}{(P \land Q) \lor (\neg R \land S)} \\
\multicolumn{2}{c}{/} & \multicolumn{2}{c}{\backslash} \\
\multicolumn{2}{c}{P \land Q} & \multicolumn{2}{c}{\neg R \land S} \\
/ & \backslash & / & \backslash \\
P & Q & \neg R & S
\end{array}
$$

Base Case

If the tree consists of the root node, then it means that $\perp$ (contradiction) is in N. This serves as the base case for the induction.

Inductive Step

Consider two sister leaves of the same parent of the tree. These leaves are labeled with L and comp(L) respectively.

Let $C1$ and $C2$ be the two false clauses at these leaves. We need to consider the following cases:

1. If some C_i does not contain L or comp(L), then C_i is also false at the parent node, completing this case.
2. Assume both $C1$ and $C2$ contain L or comp(L). Therefore, $C1 = C_1' \lor L$ and $C2 = C_2' \lor L$. If $C1$ (or $C2$) contains further occurrences of L (or C_2 of comp(L)), then the rule factoring is applied to eventually remove all additional occurrences. Therefore, eventually L is not in C_1' and $\neg L$ is not in C_2'.
3. If some C_i contains both L and comp(L), it would be a tautology, which contradicts the assumption that C_i is false at its leaf.
4. A resolution step between C_1' and C_2' on L yields $C_1' \lor C_2'$, which is false at the parent node because the resolvent neither contains L nor comp(L).

   Furthermore, the resulting tree is smaller, proving completeness by structural induction.

3 Forward chaining and Backward chaining

3.1 Forward Chaining

Forward chaining can be described logically as repeated application of modus ponens to derive new facts/-clauses.
This algorithm is linear in size of KB.

The forward chaining algorithm is linear in terms of time complexity because:

Algorithm 3 Forward Chaining Algorithm

Input: Knowledge Base (KB), Statement α
function FORWARDCHAINING(KB, α)
    Initialize agenda with known facts from the KB
    $agenda \leftarrow$ [fact for fact in KB.facts]
    Keep track of inferred facts
    $inferred\_facts \leftarrow$ set()
    while $agenda$ is not empty **do**
        Get the first fact in the agenda
        $current\_fact \leftarrow agenda[0]$
        $agenda \leftarrow$ rest of the facts in the agenda
        if $current\_fact$ is equal to α **then**
            Output: "Alpha is true!"
            return true
        end if
        if $current\_fact$ is in $inferred\_facts$ **then**
            continue
        end if
        for each rule in KB.rules **do**
            if rule applies to $current\_fact$ **then**
                $new\_fact \leftarrow$ generate new fact using the rule
                $agenda.append(new\_fact)$
            end if
        end for
        Add $current\_fact$ to the set of inferred facts
        $inferred\_facts \leftarrow inferred\_facts \cup \{current\_fact\}$
    end while
    Output: "Alpha is not provable."
    return false
end function
FORWARDCHAINING(KB, α)

- Each fact in the knowledge base is processed exactly once, either when it's initially added to the agenda or when it's derived from a rule and added to the agenda. Therefore, the number of times each fact is processed is proportional to the number of facts in the knowledge base.
- For each fact that is processed, we check if it matches the statement alpha. This is a constant-time operation and is not dependent on the size of the knowledge base.
- For each fact that is not alpha and not previously inferred, we iterate over the rules in the knowledge base. The number of iterations is proportional to the number of rules in the KB.
- When we apply a rule to a fact, generating a new fact, this is also a constant-time operation and is not dependent on the size of the knowledge base.
- The loop continues until we find alpha or until we've processed all facts. In the worst case, where alpha is not provable, we will iterate over all the facts and rules, and the algorithm terminates.

Overall, the time complexity of forward chaining is linear, O(N), where N is the total number of rules and conditions. This linearity makes forward chaining an efficient inference algorithm for rule-based systems, especially when dealing with a large number of rules and facts, as long as the rules and conditions are not overly complex.

3.2 Backward Chaining

Backward chaining is an inference method described colloquially as working backward from the goal(based on the modus ponens inference rule). It starts with a list of goals (or a hypothesis) and works backwards from the consequent to the antecedent to see if any data supports any of these consequents.

Algorithm 4 Backward Chaining Algorithm

```
Input: Knowledge Base (KB), Statement α
function BackwardChaining(α)
    if α is in KB.facts then
        Output: "Alpha is true!"
        return true
    end if
    for each rule in KB.rules do
        if conclusion of rule is α and BackwardChaining(antecedents of rule) then
            Output: "Alpha is true!"
            return true
        end if
    end for
    Output: "Alpha is not provable."
    return false
end function
BackwardChaining(α)
```

4 First Order Logic (FOL)

First-order logic uses quantified variables over non-logical objects, and allows the use of sentences that contain variables, so that rather than propositions such as "Socrates is a man", one can have expressions in the form "there exists x such that x is Socrates and x is a man", where "there exists" is a **quantifier**, while x is a variable.

4.1 Objects (Constant Symbols)

Objects, denoted by constant symbols, are the basic entities in a logical language, representing specific elements within a domain of discourse.

Example: In a logical language $\mathcal{L}$ for individuals, we may have constant symbols such as: A, B, people, numbers, colors, wars, theories, squares, pits, wumpus, etc.

These constants refer to distinct individuals within the domain.

4.2 Relations (Predicate Symbols)

Relations, expressed by predicate symbols, are properties or characteristics attributed to objects within a logical language. They yield truth values for specific combinations of objects. It can be unary relation such as: red, round, is adjacent, or n-any relation such as: the sister of, brother of, has color, comes between.

Example: In $\mathcal{L}$ for individuals, we may also have predicate symbols like:

$$\text{Married}(x, y) \text{ (indicating "} x \text{ is married to } y\text{")}$$
$$\text{Parent}(x, y) \text{ (indicating "} x \text{ is a parent of } y\text{")}$$

For instance, $\text{Married}(\text{John}, \text{Alice})$ asserts that John is married to Alice.

4.3 Functions (Functions Symbols)

Function symbols are used to represent operations or functions that take one or more arguments and produce a result. They are not based on natural language and are typically denoted by letters or symbols. For example, a function symbol might represent addition, multiplication, or other mathematical operations. For instance the function symbol "+" represents the operation of addition. For example, if we have the function symbol "+" and apply it to two arguments, x and y, it would be denoted as f(x, y) = x + y.

4.4 Variables and Quantifiers

FOL also provides variables and quantifiers which allows you to reason about collection of objects ($\neq$ Propositional Logic)

1. **Universal ($\forall$):** Universal quantifiers make assertions about all objects in our domain.
   Example - All dogs are animals.
   $$\forall x \text{ Dog}(x) \Rightarrow \text{Animal}(x)$$
2. **Existential ($\exists$):** Existential quantifiers resonate about some objects in our domain.
   Example - A flower is green.
   $$\exists x Flower(x) \wedge Green(x)$$

Note: Switching the order of universal quantifier of existential quantifiers does not alter the meaning. However, switching universal and existential quantifiers within a start changes its meaning. For instance :

- Original Statement: $\forall$ student, $\exists$ class they enjoy.
- Reordered Statement: $\exists$ class such that $\forall$ student enjoys it.
- Switched Quantifiers Statement: $\exists$ student such that $\forall$ class is enjoyed by them.

4.5 Operators ($\neg, \wedge, \vee, \doteq, \implies, \iff$)

Logical operators are symbols or word used to connect two or more expressions such that the value of the compound expression produced depends only on that of the original expressions and on the meaning of the operator.
Example - Bob has at least two friends.

$$\exists x, y\ \neg(x \doteq y) \wedge Friend(x) \wedge Friend(y)$$

Alice has exactly two friends.

$$\exists x, y\ \forall z\ Friend(x) \wedge Friend(y) \wedge (Friend(z) \implies z = x \vee z = y)$$

4.6 Inference with FOL

No self driving car is foolproof

$$\neg(\exists\ selfdriving(x) \wedge foolproof(x))$$

$$\forall\ selfdriving(x) \implies \neg foolproof(x)$$

By application of **De Morgan's Law**.