Overview

- Agenda for today:
  - Semantic argument method for proving FOL validity
  - Important properties of FOL

Motivation for semantic argument method

- So far, defined what it means for FOL formula to be valid, but how to prove validity?
- Will extend semantic argument method from PL to FOL
- Recall: In propositional logic, satisfiability and validity are dual concepts:
  \[ \neg F \text{ is unsatisfiable} \]
- Since this duality also holds in FOL, we’ll focus on validity

Semantic Argument Method to Prove Validity

- Recall: Semantic argument method is a proof by contradiction.
- Basic idea: Assume that \( F \) is not valid, i.e., there exists some \( S, \sigma \) such that \( S, \sigma \nmodels \neg F \)
- Then, apply proof rules.
- If can derive contradiction on every branch of proof, \( F \) is valid.

Proof Rules I (Review)

- All proof rules from prop. logic carry over to first-order logic.
- As before, proof rules come in pairs, for each connective, we have one case for \( \models \), one case for \( \not\models \)
- Negation elimination:
  \[
  S, \sigma \models \neg F \\
  S, \sigma \not\models F
  \]
- And elimination rule:
  \[
  S, \sigma \models F \land G \\
  S, \sigma \not\models F \\
  S, \sigma \models G
  \]
  \[
  S, \sigma \not\models F \land G \\
  S, \sigma \models \neg F \\
  S, \sigma \models \neg G
  \]

Proof Rules II (Review)

- Or elimination:
  \[
  \frac{S, \sigma \models F \lor G}{S, \sigma \models F} \quad \frac{S, \sigma \not\models F \lor G}{S, \sigma \not\models G} \
  \]
- Implication elimination:
  \[
  \frac{S, \sigma \models F \rightarrow G}{S, \sigma \not\models F} \quad \frac{S, \sigma \models F \rightarrow G}{S, \sigma \models G}
  \]
- If and only if elimination:
  \[
  \frac{S, \sigma \models F \leftrightarrow G}{S, \sigma \models F \land \neg G} \\
  \frac{S, \sigma \models F \leftrightarrow G}{S, \sigma \models \neg F \land G}
  \]
Proof Rules III (New)

- We need new rules to eliminate universal and existential quantifiers.

  - Universal elimination I:
    \[ U, I, \sigma \models \forall x. F \] (for any \( o \in U \))
    \[ U, I, \sigma[x \mapsto o] \not\models \models F \]

- Example: Suppose \( U, I, \sigma \models \forall x. \text{hates}(\text{jack}, x) \)

- Using the above proof rule, we can conclude:
  \[ U, I, \sigma[x \mapsto I(\text{jack})] \models \text{hates}(\text{jack}, x) \]

Existential Elimination Rule 1

- Existential elimination I:
  \[ U, I, \sigma \not\models \exists x. F \] (for a fresh \( o \in U \))
  \[ U, I, \sigma[x \mapsto o] \not\models F \]

  - Again, fresh means an object that has not been used before.

Proof Rules V (New)

- Finally, we need a rule for deriving for contradictions.

  - Contradiction rule:
    \[ U, I, \sigma \models p(s_1, \ldots, s_n) \]
    \[ U, I, \sigma \not\models p(t_1, \ldots, t_n) \]
    \[ (I, \sigma)[s_i] = (I, \sigma)[t_i] \text{ for all } i \in [1, n] \]
    \[ U, I, \sigma \not\models \bot \]

- Example: Suppose we have \( S, \{x \mapsto a\} \models p(x) \) and
  \( S, \{y \mapsto a\} \not\models p(y) \)

  - The proof rule for contradiction allows us to derive \( \bot \).

Universal Elimination Rule II

- Universal elimination II:
  \[ U, I, \sigma \not\models \forall x. F \] (for a fresh \( o \in U \))
  \[ U, I, \sigma[x \mapsto o] \not\models F \]

  - By a fresh object constant, we mean an object that has not been previously used in the proof.

  - Why do we have this restriction?

Example 1: Proving Validity

- Prove the validity of formula:
  \[ F : (\forall x. p(x)) \rightarrow (\forall y. p(y)) \]

  - We start by assuming it is not valid, i.e., there exists some \( S, \sigma \) such that \( S, \sigma \not\models F \).
Example 2

- Is this formula valid?
  
  \( F : (\forall x. (p(x) \lor q(x))) \rightarrow (\exists x. p(x) \lor \forall x. q(x)) \)

- Informal argument: Suppose \( \forall x. (p(x) \lor q(x)) \) holds
  
  This means either \( q(x) \) does not hold for some object \( o \), then \( p(x) \) must hold for that object \( o \) (i.e., \( \exists x. p(x) \))

Example 3

- Is this formula valid?
  
  \( F : (\forall x. (p(x, x)) \rightarrow (\exists x. \forall y. p(x, y)) \)

- How do you prove it’s not valid?
  
  Falsifying interpretation:

Example 4, cont

- Let’s now prove validity using semantic argument method
  
  \[ F : (\forall x. (p(x) \lor q(x))) \rightarrow (\exists x. p(x) \lor \forall x. q(x)) \]

- Let’s assume there is some \( S, \sigma \) that does not entail \( \phi \), and derive contradiction on all branches

Example 4

- Is the following formula valid?
  
  \( (\forall x. (p(x) \land q(x))) \rightarrow (\forall x. p(x)) \land (\forall x. q(x)) \)

Soundness and Completeness of Proof Rules

- The proof rules we used are sound and complete.

- Soundness: If every branch of semantic argument proof derives a contradiction, then \( F \) is indeed valid.

- Translation: The proof system does not reach wrong conclusions

- Completeness: If formula \( F \) is valid, then there exists a finite-length proof in which every branch derives \( \bot \)

- Translation: There are no valid first-order formulas which we cannot prove to be valid using our proof rules.
Important Properties of First Order Logic

▶ Really important result: It is undecidable whether a first-order formula is valid. (Church and Turing)

▶ Review: A problem is decidable iff there exists a procedure $P$ such that, for any input:
1. $P$ halts and says “yes” if the answer is positive
2. $P$ halts and says “no” if the answer is negative

▶ But, what about the completeness result? Doesn’t this contradict undecidability?

Semidecidability of First-Order Logic

▶ First-order logic is semidecidable

▶ A decision problem is semidecidable iff there exists a procedure $P$ such that, for any input:
1. $P$ halts and says “yes” if the answer is positive
2. $P$ may not terminate if the answer is negative

▶ Thus, there exists an algorithm that always terminates and says if any arbitrary FOL formula is valid

▶ But no algorithm is guaranteed to terminate if the FOL formula is not valid

Decidable Fragments of First-Order Logic

▶ Although full-first order logic is not decidable, there are fragments of FOL that are decidable.

▶ A fragment of FOL is a syntactically restricted subset of full FOL: e.g., no functions, or only universal quantifiers, etc.

▶ Some decidable fragments:
  ▶ Quantifier-free first order logic
  ▶ Monadic first-order logic
  ▶ Bernays-Schönfinkel class

Quantifier-Free Fragment of FOL

▶ The quantifier-free fragment of FOL is the syntactically restricted subset of FOL where formulas do not contain universal or existential quantifiers.

▶ Determining validity and satisfiability in quantifier-free FOL is decidable (NP-complete).

▶ This fragment can be reduced to a theory we will explore later, theory of equality with uninterpreted functions

Monadic First-Order Logic

▶ Pure monadic FOL: all predicates are monadic (i.e., arity 1) and no function constants.

▶ Impure monadic FOL: both monadic predicates and monadic function constants allowed

▶ Result: Monadic first-order logic is decidable (both versions)

▶ However, if we add even a single binary predicate, the logic becomes undecidable.

Bernays-Schönfinkel Class

▶ The Bernays-Schönfinkel class is a fragment of FOL where:
1. there are no function constants,
2. only formulas of the form:
$$\exists x_1, \ldots, \exists x_n, \forall y_1, \ldots, \forall y_m, F(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

▶ Result: The Bernays-Schönfinkel fragment of FOL is decidable

▶ Database query language Datalog is based on Bernays-Schönfinkel class of FOL
Datalog

- Datalog is a programming language that allows adding/querying facts in a deductive databases
- An example Datalog program:

  ```datalog
  parent(bill, mary). % Bill is Mary’s parent
  parent(mary, john). % Mary is John’s parent
  ancestor(X, Y) :- parent(X, Y).
  ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).
  ?- ancestor(X, john).
  ```

- Last statement is a query: Is there anyone in the database who is John’s ancestor (and if so, who?)

Datalog, cont.

- parent(bill, mary). % Bill is Mary’s parent
- parent(mary, john). % Mary is John’s parent
- ancestor(X, Y) :- parent(X, Y).
- ancestor(X, Z) :- parent(X, Y), ancestor(Y, Z).
- ?- ancestor(X, john).

- This program is just syntactic sugar for FOL:

  ```datalog
  parent(bill, mary) ∧ parent(mary, john) ∧
  (∀x, y. parent(x, y) → ancestor(x, y)) ∧
  (∀x, y, z. parent(x, y) ∧ parent(y, z) → ancestor(x, z)) ∧
  (∃x. ancestor(x, john))
  ```

- Thus, if this formula is satisfiable, there is someone in our database who is John’s ancestor

Datalog and Logic Programming Languages

- A Datalog interpreter is nothing more than a solver for Bernays-Schönfinkel fragment of FOL
- Since this fragment is decidable, Datalog programs always terminate
- In general, interpreters for all logic programming languages decide satisfiability in FOL or a fragment
- A popular logic programming language is Prolog
- Unlike Datalog, it is based on full FOL, so Prolog programs may not terminate

Compactness of First-Order Logic

- Another important property of FOL is compactness.
- A logic is called compact if an infinite set of sentences Γ is satisfiable if every finite subset of Γ is satisfiable.
- Theorem (due to Gödel): First-order logic is compact.

Consequences of Compactness

- Proof of compactness might look like a useless property, but it has very interesting consequences!
- Compactness can be used to show that a variety of interesting properties are not expressible in first-order logic.
- For instance, we can use compactness theorem to show that transitive closure is not expressible in first order logic.

Transitive Closure

- Given a directed graph \( G = (V, E) \), the transitive closure of \( G \) is defined as the graph \( G^+ = (V, E^+) \) where:
  
  \[ E^+ = \{(n, n') \mid \text{there is a path from vertex } n \text{ to } n'\} \]

- Observe: A binary predicate \( p(t, t') \) can be viewed as a graph containing an edge from node \( t \) to \( t' \)
- Thus, the concept of transitive closure applies to binary predicates as well
- A binary predicate \( T \) is the transitive closure of predicate \( p \) if \( \langle t_0, t_0 \rangle \in T \) if there exists some sequence \( t_0, t_1, \ldots, t_n \) such that \( \langle t_i, t_{i+1} \rangle \in p \)
### “Expressing” Transitive Closure in FOL

- **Proof I**
  - At first glance, it looks like transitive closure $T$ of binary relation $p$ is expressible in FOL:
    \[ \forall x, \forall y, (T(x, z) \iff (p(x, z) \lor \exists y, p(x, y) \land T(y, z))) \]
  - But this formula does not describe transitive closure at all!
  - To see why, consider $U = \mathbb{N}$, $p$ is equality predicate, and $T$ is relation that is true for any number $x, y$.
  - Clearly, this $T$ is not the transitive closure of equality, but this structure is actually a model of the formula.
  - Thus, the formula above is not a definition of transitive closure at all!

- **Proof II**
  - Recall: $\Gamma$ is a set of propositions encoding $T$ is transitive closure of $p$.
  - Now, construct $\Gamma'$ as follows:
    \[ \Gamma' = \Gamma \cup \{ T(a, b), \Psi^1, \Psi^2, \Psi^3, \ldots \} \]
  - **Observe:** $\Gamma'$ is unsatisfiable because:
    1. Since $\Gamma$ encodes that $T$ is transitive closure of $p$, $T(a, b)$ says there is some path from $a$ to $b$
    2. The infinite set of propositions $\Psi^1, \Psi^2, \ldots$ say that there is no path of any length from $a$ to $b$

- **Proof III**
  - Now, consider any finite subset of $\Gamma'$:
    \[ \Gamma'' = \Gamma \cup \{ T(a, b), \Psi^1, \Psi^2, \Psi^3, \ldots \} \]
  - Clearly, any finite subset does not contain $\Psi_i$ for some $i$.
  - **Observe:** This finite subset is satisfied by a model where there is a path of length $i$ from $a$ to $b$
  - Thus, every finite subset of $\Gamma''$ is satisfiable.
  - By the compactness theorem, this would imply $\Gamma'$ is also satisfiable
  - But we just showed that $\Gamma'$ is unsatisfiable!
  - Thus, transitive closure cannot be expressed in FOL!