St. Francis Xavier University
Department of Computer Science

CSCI 544: Computational Logic
Group Lecture and Report
Fall 2025

The group lecture and report are major assessments of this course. These assessments consist of three components: a lesson plan, the lecture itself, and an individual report. The group lecture is preferably completed in a group of four students, though *very limited* exceptions can be made if a group of four cannot be formed.

The lesson plan is worth a total of 10% of your final grade. The lecture component is worth a total of 20% of your final grade. The individual report is worth a total of 10% of your final grade. You must complete all of these components in order to pass the course.

Do not leave these assessments until the last minute! Start as soon as possible!

Lesson Plan

Due October 22, 2025 in lecture

The lesson plan is a short document meant to establish your group and your group's chosen lecture topic.

The lesson plan should include, at a minimum, the following information:

- The names of your group members
- One or two paragraphs giving a high-level introduction to your group's lecture topic
- A description of what each group member plans to contribute to the lecture
- A broad outline of the structure of your group's lecture (e.g., X minutes for introduction, Y minutes for background, etc.)
- A list of preliminary references that your group will use to create your lecture materials

A title page is not required.

When planning what each group member can contribute to the lecture, consider splitting the lecture topic into independent sections or subtopics and assigning one section or subtopic to each group member.

Existing survey articles or books are great preliminary references, as they often include a large amount of background information presented at an accessible level. If your group needs pointers to such preliminary references, let me know and I can try to help.

All references must be cited in a consistent style (e.g., IEEE citation style). Citations to articles and books are preferred over citations to lecture notes, Wikipedia, or websites.

Your group's lesson plan should be a 1–2 page double-spaced document written in 12-point text. This is a strict page limit, but references do not count toward the limit. The lesson plan will be marked in terms of completeness, organization, and adherence to guidelines.

Each group only needs to submit one lesson plan for all group members.

Lecture

Due date varies (last ∼3 weeks of lectures)

The lecture is a ∼50 minute presentation on your group's chosen topic. Your group will not be presenting for the entire length of time; you should budget ∼5 minutes for setup at the beginning and ∼5 minutes for questions and teardown at the end. Therefore, your actual lecture time will be closer to ∼40 minutes, and each group member should spend an approximately equal amount of time speaking (i.e., in a group of four, each group member speaks for ∼10 minutes).

Group members should take turns presenting so that no one is unfairly burdened with having to speak for the entire duration. This is an opportunity for individuals to use their strengths; for example, one person may cover the introduction, one person may work through an example, one person may choose to lead a live coding session, etc.

Your group's lecture should be presented at a level similar to other lectures in this course: targeted at a graduate student audience that is familiar with computational logic. Your group can refer to past lecture notes as background material, but your group should ultimately develop your own notes or slides for use in the lecture itself.

Your group must submit your lecture materials (e.g., notes, slides, etc.) *at least one day before* the lecture is scheduled. Your lecture will be marked in terms of three broad categories: quality (e.g., focused on topic, targeted at an appropriate level), accuracy (e.g., correct summary of main results, material all related to topic), and organization (e.g., notes/slides are clearly written, lecture is well-paced).

All lecture material must be appropriately referenced, and citations must be included wherever necessary.

Lastly, remember that *quality is better than quantity.* A very well-produced lecture that takes ∼30 minutes is better to experience than a poorly planned lecture that takes ∼40 minutes.

Report

Due December 5, 2025 in lecture

The report is an individual submission meant to complement the group lecture. In this report, you will write a brief survey of your lecture topic *in your own words* and give a short reflection on your group lecture experience.

On the first page of your report, you should include, at a minimum, the following information:

- The names of your group members and the topic of your lecture
- A list of the specific contributions you made in your group
- Approximately one paragraph discussing aspects of your lecture that you feel went well and aspects of your lecture that you feel could have been improved. Your opinion will not be shared with other group members.

In the rest of your report, you should include the following information:

- 3–4 pages summarizing your lecture topic, written in the style of a survey article or short lecture notes
- A complete list of references used in your report (not in your group lecture in general)

You must additionally include a title page that lists your name, report topic, and course details.

Note that your 3–4 page summary should be written *in your own words*; you must submit your own summary, and not one written collaboratively by all group members. If your group divided work by assigning specific sections or subtopics to individual members, this is a good opportunity to showcase your knowledge about the particular section or subtopic you focused on while designing your group lecture.

All references must be cited in a consistent style (e.g., IEEE citation style). Citations to articles and books are preferred over citations to lecture notes, Wikipedia, or websites.

Your report should be a 4–5 page double-spaced document written in 12-point text. This is a strict page limit, but references and the title page do not count toward the limit. The report will be marked primarily in terms of completeness, organization, and adherence to guidelines.

Since the report is due at the end of the term, the report deadline is firm. Late reports will not be accepted.

Suggested Topics

Below, I offer a selection of topics that your group can choose for your lecture. If your group wishes to study and present a topic related to this course that is not on this list, your group must check with me first to approve the proposed topic.

- **Special topics**
  - **Binary decision diagrams (BDDs).**
    BDDs are data structures used to represent Boolean functions in a compact and canonical way. By reducing equivalent logical substructures and enforcing an ordering on variables, BDDs allow efficient manipulation of large Boolean formulas, which is especially useful in hardware design and verification.
  - **Gentzen calculus.**
    The Gentzen calculus, or sequent calculus, is a formal system to study logical proofs. Proofs are constructed as sequences of formulas with inference rules that emphasize structural properties of logic. This framework is foundational for proof theory, enabling results like cut elimination and providing insights into the structure of mathematical reasoning.
  - **Hilbert calculus.**
    The Hilbert-style calculus is built around a small set of axioms and inference rules. Unlike sequent calculus, proofs in Hilbert systems tend to be longer and less intuitive, but they are minimal and elegant. This style of system has influenced modern formal logic deeply, serving as a benchmark for what it means for a set of axioms to be complete and sound.
  - **Peano arithmetic.**
    Peano arithmetic formalizes the set of natural numbers using axioms. It provides rules for zero, successor functions, and induction. It is a central subject in mathematical logic because it embodies the foundations of number theory, but it is also known to be incomplete due to Gödel's incompleteness theorems, showing the limits of formal reasoning.
  - **Predicate logic with equality.**
    Just like how predicate logic extends propositional logic with quantifiers and variables, adding equality allows explicit reasoning about when two terms denote the same object. This extension is critical in computer science, as it underpins reasoning about data structures, algebraic properties, and automated theorem proving.
- **Advanced logic systems**
  - **Belief logic.**
    Belief logic is used to reason about knowledge and belief states of agents. It introduces modal operators like "knows" and "believes" to capture reasoning about information. This is central to AI, distributed systems, and game theory, where modeling the information and beliefs of different agents is crucial for coordination and strategy.
  - **Deontic and imperative logic.**
    Deontic logic formalizes concepts like obligation, permission, and prohibition, while imperative logic deals with commands and prescriptions rather than truth statements. These systems are important in ethics, legal reasoning, and computer science applications like policy specification, where normative rules must be modeled and enforced logically.
  - **Modal logic.**
    Modal logic generalizes classical logic by introducing operators like "necessarily" and "possibly". Different modal systems model different notions, such as time, obligation, or knowledge. It has a rich theory connecting syntax with semantics, making it one of the most widely applied non-classical logics in philosophy, computer science, and linguistics.

- **Temporal logic.**
  Temporal logic allows reasoning about propositions that change over time, using operators like "always", "eventually", and "until". It is used in program verification and model checking, where correctness depends on sequences of states rather than static properties, such as ensuring a safety condition always holds or that a request is eventually granted.

- **Program verification and model checking**
  - **Hoare logic.**
    Hoare logic provides a formal system for reasoning about the correctness of computer programs using Hoare triples of the form `{P} C {Q}`, where `P` and `Q` are assertions before and after executing a command `C`. It is a cornerstone of program verification, giving a structured way to prove properties of imperative programs systematically.
  - **Linear-time temporal logic.**
    Linear-time temporal logic, or LTL, is a form of temporal logic where properties are evaluated along linear sequences of states. It is widely used in model checking for specifying safety and liveness properties. Its syntax balances expressive power with decidability, making it practical for automated verification of software and hardware systems.
  - **Software verification, partial correctness, and total correctness.**
    Partial correctness ensures that if a program terminates, then its result satisfies the specification; while total correctness adds the requirement of termination itself. These notions are central to verifying reliability in software, as they address not only whether programs produce the right results but also whether they do so in finite time.

- **Proof assistants and theorem provers**
  - **The Coq proof assistant.**
    Coq is an interactive theorem prover based on the calculus of inductive constructions. It allows users to write mathematical definitions, state theorems, and develop machine-checked proofs. It has been used in both pure mathematics (e.g., proving the four-color theorem) and computer science (e.g., verifying compilers).
  - **The HOL/Isabelle proof assistants.**
    HOL and Isabelle are proof assistants based on higher-order logic. They emphasize extensibility and flexibility, making them suitable for a wide range of applications in mathematics and computer science. Isabelle supports multiple logics through its meta-logic framework, while HOL systems are optimized for classical higher-order reasoning.
  - **The Lean proof assistant.**
    Lean is a relatively new proof assistant designed for both formal mathematics and verification. It emphasizes clean design, automation, and integration with modern programming practices. It has gained popularity in the mathematical community through the Lean mathematical library.

- **Logic programming languages and software tools**
  - **Agda.**
    Agda is both a programming language and proof assistant based on dependent types. It allows writing programs where types can encode rich logical properties, making correctness part of the code itself.
  - **Prolog.**
    Prolog is a classic logic programming language where computation is performed by querying a database of facts and rules. Its execution model is based on resolution and unification.
  - **SAT solvers.**
    SAT solvers are algorithms that determine whether a Boolean formula is satisfiable. Modern SAT solvers are highly optimized and can handle formulas with millions of variables. They have become essential tools in verification, optimization, planning, and even cryptanalysis, often serving as back-end engines for more complex reasoning systems.