INSTRUCTION DIVISION
FIRST SEMESTER 2018-2019
Course Handout Part II

In addition to part-I (General Handout for all courses appended to the time table) this portion gives further specific details regarding the course

Course No. : **CS / IS F214**
Course Title : **Logic in Computer Science**

Instructor-in-Charge: **Shan Sundar Balasubramaniam (email: sundarb)**
Instructors:
Shan Sundar Balasubramaniam (email: sundarb) – Tut. Sections 1 & 3
Jagat Sesh Chella (email: jagatsesh) – Tut. Sections 2 & 4

1. a. Scope:

The purpose of this course is to introduce the formal study of Logic to Computer Science undergraduates. Within this context the course introduces ***first order propositional and predicate logics*** as well as special topics such as ***modeling***, ***program verification***, and ***temporal logics***.

Natural Deduction as a proof system for propositional and predicate logics is covered. ***Soundness*** and ***Completeness*** of proofs are covered but only for propositional logic. The course also covers some applications in modeling computing systems and reasoning about programs - in particular, ***model checking*** based on temporal logics and program verification using ***Floyd-Hoare logic***. The relationship between formal logic and pragmatics of computing is highlighted via a few specific topics: (i) the **satisfiability** problem in propositional logic and its computational complexity (ii) ***Horn-clause problem solving*** as a basis for programming.

b. Course Objectives

On completion of this course the student shall be able to

(i) write statements and proofs in first order predicate logic,
(ii) explain the limitations of first order predicate logic and identify the need for higher-order logics or temporal logics in specific contexts
(iii) state and argue soundness and completeness properties of logics
(iv) write grammars for the syntax of logics
(v) write algorithms for verifying satisfiability / validity where applicable

(vi) relate problems in logic to problems in computation
(vii) use logics to formally specify and verify computational properties.

2. Text Book:

T1: Michael Huth and Mark Ryan. *Logic in Computer Science – Modelling and Reasoning about Systems*. Cambridge University Press. 2nd Edition. 2004.

3. Course Plan:

3.a. Modules

Module #	Topics
I	**Introduction to Logics and Proofs. Overview of the interplay between Logic and Computing.**
II	**Propositional Logic: Natural Deduction, Syntax and Semantics, Soundness and Completeness, Satisfiability and Validity – Forms and Algorithms.**
III	**Predicate Logic: Syntax and Semantics, Logic Programming, Natural Deduction, Limitations of First Order Logic.**
IV	**Program Verification: Floyd-Hoare Logic – Pre-conditions, Post-conditions, and Loop Invariants; Verification of imperative programs – Partial and Total correctness.**
V	**Temporal Logic: LTL, CTL, and Model Checking**

3. b. Lecture Schedule:

Lecture #	Module #	Topic	Learning Outcome(s) [The student will be able to:]	Reading
L1	I.a	Why study Logic?	• state a few reasons to study logics and proofs	-
L1	I.b	A broad and selective History of Logic and Proofs (**HLP**) - Part I: Euclid's Formalization of Geometry. **Hilbert's Program** and the Formalization	• state typical issues and debates in formalizing proofs (and mathematical arguments in general) • state the distinction between	-

		problem. Russell's efforts in formalization and **Russel's paradox**.	axioms, statements, and proofs	
L2	**I.c**	HLP – Part II: **Brouwer's intuitionism** – Intuitionist's Critique on **Proof by Contradiction** and (the use of) **Law of Excluded Middle**	• distinguish between a **constructive proof** and a **non-constructive proof** • state the intuitionist position / critique on non-constructive proofs	
L2	**I.d**	HLP – Part III: **Godel's Incompleteness** result and its impact on the formalization of mathematics	• state soundness and completeness requirements of proof systems and • explain – at a high level – the implications of Godel's incompleteness result	-
L2	**I.e**	Logic and Computing (**L&C**) – Part I: **Godel**, **Church**, and **Turing**: **Computability** – Systems for defining computability; **Church-Turing Thesis**.	• informally define the notion of "computability" • state the equivalence of schemes / mechanisms for computability	-
L3	**I.f**	**L&C** – Part II: Logic and Computing: **Non-computable Problems** – Example: **Halting**	• state the **Halting Problem** and argue that it is not computable	
L4	**I.g**	**L&C** – Part III: **Time Complexity**, **Complexity Classes**, Is **P = NP**?	• define complexity classes **P** and **NP** • state the implications of **P=NP** or **P!=NP**	-
L4	**I.h**	**L&C** – Part IV: Boolean **Satisfiability** (**SAT**) and **Horn-Clause Satisfiability** problems; their relationship to the classes **NP** and **P**;	• verify satisfiability of Boolean expressions and Horn-style expressions.	-
L4	**I.i**	**L&C** – Part V: Logic and Programming: **Logic Programming** and **Prolog**.	• describe the relation between logic and programming at a high level	-
L5	**II.a**	**Propositional Logic**: **Natural Deduction** as a Proof System.	• explain the notational conventions used in Natural Deduction	T1 Sec. 1.2.1
L5	**II. b**	Propositional Logic: Natural Deduction: **Conjunction** Rules	• apply proof rules involving conjunction	T1 Sec. 1.2.1
L5	**II.c**	Propositional Logic: Natural Deduction: **Implication** Rules	• apply ***modus ponens*** and implication introduction	T1 Sec. 1.2.1
L6	**II.d**	Propositional Logic: Natural Deduction: **Disjunction** Rules	• apply proof rules involving disjunction	T1 Sec. 1.2.1

L6	**II.e**	Propositional Logic: Natural Deduction: **Negation** Rules	• apply proof rules involving negation	T1 Sec. 1.2.1
L6	**II.f**	Propositional Logic: Natural Deduction: **Double Negation** Rules	• apply proof rules involving double negation	T1 Sec. 1.2.1
L6	**II.g**	Propositional Logic: Natural Deduction: Derived Rules: ***Modus Tollens, Law of Excluded Middle*** *(**LEM**),* ***Proof by Contradiction*** *(**PbC**).*	• apply ***modus tollens***, **LEM**, and **PbC** *in proofs* • derive **modus tollens**, **LEM**, and **PbC** from basic proof rules	T1 Sec. 1.2.2
L7	**II.h**	Syntax: **Context Free Grammars** (**CFG**): Notation and Examples	• define **CFG**s for simple constructs	T1 Sec. 1.3
L7	**II.i**	Context Free Grammars: **Parse Trees**: Examples	• illustrate and explain the internal structure of context-free constructs	T1 Sec. 1.3
L8	**II.j**	Propositional Logic: Syntax: Well-formed-formulas and grammar.	• state the formal syntax of propositional logic as a CFG	T1 Sec. 1.3
L8	**II.k**	Propositional Logic: Syntax: Different forms of grammar: **Precedence** and **Order of evaluation**: Rules to handle precedence and over-ride precedence	• write grammar rules to define the syntax of propositional logic with a specific order of evaluation	T1 Sec. 1.3
L9	**I.j**	Proof Techniques: **Mathematical Induction**	• articulate the generic structure / template of proofs by mathematical induction	-
L9	**I.k**	Proof Techniques: **Structural Induction** and Examples	• relate mathematical induction to structural induction	-
L9	**I.l**	Grammars, Parse Trees and Structural Induction – Approach	• illustrate the relation between languages and inductive proofs • explain structural induction as a proof technique	T1 Sec. 1.4.2
L9	**I.m**	Structural Induction - Examples	• write proofs using structural induction	T1 Sec. 1.4.2
L10	**II.l**	Propositional Logic: Semantics: Truth Tables, **Soundness** and **Completeness**	• interpret sentences in propositional logic • state the soundness and completeness results w.r.t. propositional logic	T1 Sec. 1.4.1
L10	**II.m**	Propositional Logic: Semantics: Equivalence and **Normal Forms**	• argue or verify the equivalence (or the lack of it) of given propositional	T1 Sec. 1.4.1

			formulas	
L10	**II.n**	Propositional Logic: Syntax: Grammars for Normal Forms	• define a grammar for a given normal form	-
L11	**II.o**	Propositional Logic : Soundness	• state soundness arguments in general and soundness arguments for logics in particular	T1 Sec. 1.4.3
L11	**II.p**	Propositional Logic: Soundness Proof	• argue that proofs in propositional logic are sound	T1 Sec. 1.4.3
L12	**II.q**	Propositional Logic: Completeness	• state completeness arguments in general and completeness arguments for logics in particular	T1 Sec. 1.4.4
L12	**II.r**	Propositional Logic: Completeness Proof Overview	• state completeness arguments for propositional logic	T1 Sec. 1.4.4
L12	**II.s**	Propositional Logic: Completeness Proof	• argue that propositional logic is complete using structural induction	T1 Sec. 1.4.4
L13	**II.t**	Propositional Logic: **Conjunctive Normal Form** (**CNF**) and **Disjunctive Normal Form** (**DNF**)	• recognize propositional formulas in **CNF** / **DNF** • rewrite propositional formulas in **CNF** / **DNF**	T1 Sec. 1.5.2
L13	**II.u**	Propositional Logic: **Validity**	• verify whether a propositional formula is valid or not	T1 Sec. 1.5.1
L13	**II.v**	Propositional Logic: Validity of specific forms (**CNF**), Algorithm(s) for validity	• verify validity of a formula in CNF • write a program to verify validity of a formula in CNF	T1 Sec. 1.5.1
L13	**II.w**	Propositional Logic: Validity and **Satisifiability**; Satisfiability of specific forms (**DNF**)	• verify whether a propositional formula is satisfiable or not • state the relation between validity and satisifiability of propositional formulas	T1 Sec. 1.5.1
L14	**II.x**	Propositional Logic: **Horn Clauses** and **Horn Formulas**	• distinguish Horn formulas from general formulas in propositional logic • identify when a propositional formula can be rewritten in Horn form	T1 Sec. 1.5.3

L14	**II.y**	Propositional Logic: Satisifiability of Horn Formulas. Algorithm for Satisfiability of Horn Formulas	• verify whether a Horn formula is satisfiable or not • write a program to verify validity of a Horn formula	T1 Sec. 1.5.3
L14	**II.z**	Propositional Logic: Algorithm for Satisfiability of Horn Formulas	• write a program to verify validity of a Horn formula	T1 Sec. 1.5.3
L15	**III.a**	Expressiveness of Propositional Logic – Limitations: Need for **predicates**	• illustrate statements and arguments that cannot be expressed in propositional logic • explain why predicates are more expressive than propositions	T1 Sec. 2.1
L15	**III.b**	Expressiveness of Propositional Logic – Limitations: Need for **variables** and **quantification**	• illustrate statements that cannot be expressed in propositional logic • explain how variables and quantification add expressive power	T1 Sec. 2.1
L16	**III.c**	Introduction to **Predicate Logic**: Predicates vs. Functions and Need for **Function Terms**	• distinguish between predicates and functions • explain when function terms are more expressive than predicates	T1 Sec. 2.1
L16	**III.d**	Predicate Logic: Features and Examples	• write formulas in predicate logic	T1 Sec. 2.2
L17	**III.e**	Predicate Logic: Formal Syntax: Grammars and Parse Trees	• write grammar rules for defining terms • write grammar rules for defining formulas in Predicate Logic • use the grammar rules to parse formulas in Predicate Logic	T1 Sec. 2.2.1 to 2.2.2
L17	**III.f**	Predicate Logic: **Theorem Proving**: Proof Steps; Automating the proof steps.	• explain how proofs in predicate logic can be approached. • state issues in automating proofs steps	-
L17	**III.g**	Horn Clause Programming: Theorem Proving and **Logic Programming**, Programming with Horn Clauses	• explain the relation between Theorem Proving and Logic Programming • state how Horn Clauses can be used for programming	-

L18	**III.h**	Logic Programming and **Prolog** : Proof Steps and **Term Unification**.	• perform unification of terms	-
L18	**III.i**	Logic Programming and Prolog: **Proof Search**, **Backtracking**	• apply search and backtracking to execute a Prolog program	
L18	**III.j**	Programming in Prolog: Examples – Facts and Rules; Queries	• write simple programs in Prolog	-
L19	**III.j**	Programming in Prolog: **Inductive / Recursive definitions, Lists and Other Data Structures**	• write programs using recursion in Prolog	-
L20	**III.k**	Programming in Prolog: Problem Solving in Prolog	• write reasonably large programs to solve complex problems using Prolog	-
L21	**III.l**	**Predicate Logic**: Proofs and Proof Rules: Introduction: Need for a **Substitution** Operation	• explain / illustrate the need for a substitution operation in applying proof rules of Predicate Logic	T1 Sec. 2.2
L21	**III.m**	Predicate Logic: Syntax: **Free Variables** and **Bound Variables**	• identify and distinguish between free and bound variables in Predicate Logic formulas	T1 Sec. 2.2.3
L21	**III.n**	Predicate Logic: Substitution: Definition and Examples	• apply substitution on Predicate Logic formulas	T1 Sec. 2.2.4
L22	**III.o**	Predicate Logic: Proof Rules: Equality	• explain the need for rules for equality • use the rules for equality	T1 2.3.1
L22	**III.p**	Predicate Logic: Natural Deduction: Rules for **Universal Quantification**	• apply rules for universal quantification in proofs of predicate logic formulas	T1 Sec. 2.3.1
L22	**III.q**	Predicate Logic: Natural Deduction: Rules for **Existential Quantification**	• apply rules for existential quantification in proofs of predicate logic formulas	T1 Sec. 2.3.1
Self Study	**III.s**	Predicate Logic: Syntactic Equivalence	• deduce equivalences of formulas in predicate logic	T1 Sec. 2.3.2

L23	**III.r**	Predicate Logic: Introduction to Semantics: **Models and Interpretation** – Examples	• formally define models for different theories • interpret formulas in predicate logic according to given models	T1 Sec. 2.4.1
L24	**III.s**	Predicate Logic: Semantics: The **Model-Checks** Relation	• apply the model-checks relation on predicate logic formulas	T1 Sec. 2.4.1
L25	**III.t**	Predicate Logic: Semantics: **Semantic Entailment**	• illustrate and explain semantic entailment in predicate logic	T1 Sec. 2.4.2
L25	**III.u**	Predicate Logic: Semantics: Validity and Satisifiability	• state and explain validity and satisfiability in Predicate Logic • explain and illustrate issues in verifying satisfiability and validity of formulas in Predicate Logic	T1 Sec. 2.4.3
L25	**III.v**	Predicate Logic: Soundness and Completeness	• state and explain claims regarding Soundness and Completeness of Predicate Logic	-
L26	**I.n**	Proof Techniques: **Cantor's Diagonalization –** Examples	• articulate Cantor's diagonalization technique and • apply the same on simple examples	-
L27	**I.o**	Review: **Undecidability and the Halting Problem**	• state and explain the notion of undecidability • state and explain the Halting problem. • explain the use of diagonalization in proving the undecidability of Halting problem	-
L27	**III.w**	Predicate Logic: *Validity is Undecidable*: Proof sketch using Diagonalization	• provide an proof argument for why *Validity in Predicate Logic is undecidable* using diagonalization	-
L28	**III.w**	Proof Techniques: Reduction and the use of Reduction to prove computability	• explain the concept of reduction (between problems) • apply reduction to derive simple undecidability proofs of problems	
L28	**III.w**	Predicate Logic: ***Validity is***	• provide an explanation for why	T1 Sec.

		Undecidable: Proof approach using Reduction	*Validity in Predicate Logic is undecidable* using the idea of a reduction to a known undecidable problem	2.5
L29	**III.y**	Predicate Logic: **Expressiveness**: Inexpressible Properties: Example and Proof.	• provide examples of statements that cannot be expressed in First Order Predicate Logic • provide an intuitive argument as to why such statements are inexpressible in First Order Predicate Logic • provide a rigorous argument as to why *reachability* in graphs is inexpressible in First Order Predicate Logic	T1 Sec. 2.6
L29	**III.z**	**Existential Second Order Logic**: Expressiveness: Examples(s)	• provide examples of statements that require existential quantification over predicates	T1 Sec. 2.6.1
L29	**III.aa**	**Universal Second Order Logic**: Expressiveness: Example(s)	• provide examples of statements that require universal quantification over predicates	T1 Sec. 2.6.2
L30	**IV.a**	Program Verification: **Floyd-Hoare Logic**: ***Pre-conditions*** and ***Post-Conditions***	• describe what pre-conditions and post-conditions are	T1 Sec. 4.2.2
L30	**IV.b**	Program Verification: Floyd-Hoare Logic: **Rules for Assignment and Sequencing**	• write pre-conditions and post-conditions for assignment statements • compose pre-conditions and post-conditions over sequential statements	T1 Sec. 4.2.2 & 4.3.1
L31	**IV.b**	Program Verification: Floyd-Hoare Logic: **Rule for Conditional Statements**	• compose pre-conditions and post-conditions over conditional statements	T1 Sec. 4.2.2 & 4.3.1
L31	**IV.c**	Program Verification: Floyd-Hoare	-	T1 Sec. 4.2.2 &

		Logic: **Meta-Rules**		4.3.1
L31	**IV.d**	Program Verification: **Verification of Straight-Line programs** - Examples	• apply rules of Floyd-Hoare Logic to verify properties of straight line programs	T1 Sec. 4.3.1
L32	**IV.e**	Program Verification: Floyd-Hoare Logic: **Program Variables** and **Logical Variables**	• identify and distinguish between Program Variables and Logical Variables in verification using Floyd-Hoare Logic • illustrate the need for Logical Variables with examples	T1 Sec. 4.3.1
L32	**IV.f**	Program Verification: **Partial Correctness** and **Total Correctness**	• distinguish between partial correctness arguments and total correctness arguments • write termination proofs	T1 Sec. 4.3.1
L32	**IV.g**	Program Verification: Floyd-Hoare Logic: **Loop Invariants**	• illustrate and explain what loop invariants are	T1 Sec. 4.2.2 & 4.3.1
L32	**IV.h**	Program Verification: Floyd-Hoare Logic: Verifying correctness of Loops: **Partial Correctness using Invariants** – Examples	• write loop invariants given simple loops • provide correctness arguments using loop invariants	T1 Sec. 4.2.2 & 4.3.1
L33	**IV.i**	Program Verification: Floyd-Hoare Logic: Proof Rules and Verification Examples	• provide partial correctness proofs for small programs	T1 Sec. 4.3.1 & 4.3.2
L34	**IV.j**	Program Verification: Case Study	• apply Floyd-Hoare logic to verify properties of reasonably large programs	-
L35	**IV.k**	Program Verification: Approaches and Limitations	• describe issues and limitations in proving correctness of programs	T1 Sec. 3.1
L35	**V.a**	**Introduction to Dynamic Logics**	• provide examples where “time” needs to be specified explicitly in logical formulas and the form in	T1 Sec. 3.1

			which it needs to be specified	
L35	**V.b**	Software Modeling: **State Machines**	• illustrate and explain specification using state machines	T1 Sec. 2.7.1
L36	**V.c**	Software Modeling: State Machines as Models and **Model Checking**	• describe model checking • define state machines for simple problems / computations	T1 Sec. 2.7.1
L36	**V.d**	**Linear-time Temporal Logic** (**LTL**) – Introduction	• describe LTL	T1 Sec. 3.2
L37	**V.e**	**LTL**: Syntax and Formulas	• provide examples of formulas in LTL • illustrate internal structure of formulas in LTL	T1 Sec. 3.2.1
L37	**V.f**	**LTL**: Semantics: Transitions and Paths	• interpret LTL formulas	T1 Sec. 3.2.2
L38	**V.g**	**LTL**: Specifications: Examples	• write LTL specifications for complex scenarios	T1 Sec. 3.2.3
L39	**V.h**	Properties expressible in Temporal Logics – Examples	• define the problem of model checking for a particular case	T1 Sec. 3.3.1
L39	**V.i**	Limitations of Linear Time and LTL:	• state and illustrate the limitations of LTL	T1 Sec. 3.4
L40	**V.j**	**Branching Time** and **Branching Time Temporal Logic** (**CTL**) - Introduction	• read, write, and explain CTL formulas	T1 Sec. 3.4
L40	**V.k**	**CTL** – Semantics of Temporal Operators, Examples.	• interpret CTL formulas • differentiate between interpretation in LTL and that in CTL	T1 Sec. 3.4
L41	**V.l**	**Model Checking in LTL/CTL**	• perform model checking of properties stated in CTL/LTL	T1 Sec. 3.6.1
L42	**-**	Course Summary	-	-

4. Evaluation

4. a. Evaluation Scheme:

Component	Weight	Date	Remarks
Quizzes (3)	(3 x 20M =) 60M	In Tutorial Sessions	Open Book (scheduled on one-class notice)
Assignment	40M	in Oct. & Nov.	Take Home
Mid-Term Test (90 minutes)	43M	9/10 2:00 - 3:30 PM	(scheduled centrally) Open Book
Comprehensive Exam (120 minutes)	57M	3/12 FN	(scheduled centrally) Open Book
TOTAL	200M	-	-

4. b. Make-up Policy:

- No Make-up will be available for the Assignment under any condition.
- Late submission of an assignment will incur a penalty of 25% per 24 hours after the deadline.
- There will be one make-up (for all three quizzes put together) i.e. a student can take a make-up for at most one quiz. The make-up quiz will be conducted after all the regular quizzes and the coverage for that will be announced later. ***It is the responsibility of the student to request for a make-up – i.e. a make-up is not automatic.***
- Prior Permission of the Instructor is usually required to get a make-up for the mid-term test.
- Prior Permission of (Associate) Dean, Instruction is usually required to get a make-up for the comprehensive exam.
- A make-up shall be granted only in genuine cases where - *in the Instructor's / Dean's judgment* - the student would be physically unable to appear for the quiz/test/exam. Instructor's / Dean's decision in this matter would be final.

4.c. Fairness Policy:

- Student teams are expected to work on their own on assignments.
- All students are expected to contribute equally within a team. The instructor's assessment regarding the contributions of team members would be final.
- **Any use of unfair means in quizzes, assignment, or test/exam will be reported to the Unfair means committee and will be subject to the severest penalty applicable:**
  - Unfair means would include copying from other students or from the Web or from other sources of information including electronic devices.
  - All parties involved would be treated equally responsible: *allowing others to copy one's work is enabling unfair means and is equally un-acceptable*.

5. Consultation / Office Hours:

- **Mondays** and **Wednesdays: 12.15pm to 1.15pm** and **Thursdays: 2.30pm to 3.30pm** in WILP office (1st floor, Library Building, Rear-entrance).
- ***Or by appointment via email***

6. Contents and Notices: All lecture slides will be posted on the course website on Nalanda. Notices concerning this course will be displayed online (on Nalanda) only. If there is a need, email would be used on short notice (12 hours) and only BITS Pilani mail would be used.

Instructor-In- Charge, CS F214.