University of St Andrews

MARTINMAS 2022-23 EXAMINATION DIET
SCHOOL OF COMPUTER SCIENCE

MODULE CODE: **CS3050**

MODULE TITLE: Logic and Reasoning

TIME TO HAND IN: 3 hours

EXAM INSTRUCTIONS: a. Answer all three questions
b. Each question carries 20 marks

This assessment consists of exam-style questions and you should answer as you would in an exam. You cannot copy or paraphrase text or material from other sources and present this as your own work. Your exam answers should be entirely your own work without unacknowledged input from others. If you are in any doubt, you should clearly acknowledge the origin of any material, text passages or ideas presented (e.g. through references). You must not co-operate with any other person when completing the exam, which must be entirely your own work. You must not share any information about the exam with another person (e.g. another student) or act on any such information you may receive. Any attempt to do so will be dealt with under the University's Policy for Good Academic Practice and may result in severe sanctions. You must submit your completed assessment on MMS within 3 hours of you downloading the exam. Assuming you have revised the module contents beforehand, answering the questions should take no more than three hours.
Some question may have word limits. These will be stated at the start of the question (or part question) and may be mandatory or advisory.

Answers which exceed a **mandatory** word limit may be penalised at the rate of 5% of the available marks for being overlength and a further 5% for each 10% over the word limit. So if the limit on a 20 mark question was 1000 words, an answer of 1201 words would attract a 3 mark penalty.

An **advisory** word limit is a guide to the level of detail and amount of information expected in an answer. Longer answers may lose marks for including large amounts of irrelevant material, or for failing to state arguments clearly and concisely.

Additional Instructions

Please note that in questions requiring a proof, you must show working and reasoning for full credit. Partial credit will be available for attempts that do not succeed in achieving a complete proof, as long as some understanding is demonstrated.

1. (a) **Advisory Word Limit: 250 Words in total for (a)**
Explain the difference between syntactic and semantic entailment for propositional logic. In your answer, you should also explain how syntactic and semantic entailment relate to soundness and completeness. [4 marks]

(b) Prove the following using the rules of natural deduction shown in Table 1.

(i) Prove $\neg(P \wedge Q) \Leftrightarrow \neg P \vee \neg Q$. In other words, prove $\neg(P \wedge Q) \vdash \neg P \vee \neg Q$ and $\neg P \vee \neg Q \vdash \neg(P \wedge Q)$ [8 marks]

(ii) Prove $\exists x.(P(x) \vee Q(x)) \vdash (\exists x.P(x)) \vee (\exists x.Q(x))$ [8 marks]

[Total marks 20]

And elimination: $\dfrac{\mathcal{X} \wedge \mathcal{Y}}{\mathcal{X}} \wedge E \quad \dfrac{\mathcal{X} \wedge \mathcal{Y}}{\mathcal{Y}} \wedge E$	True introduction: $\dfrac{}{\boldsymbol{T}} \boldsymbol{T}I$
And introduction: $\dfrac{\begin{array}{c}\mathcal{X} \\ \mathcal{Y}\end{array}}{\mathcal{X} \wedge \mathcal{Y}} \wedge I$	Implication elimination: $\dfrac{\begin{array}{c}\mathcal{X} \\ \mathcal{X} \to \mathcal{Y}\end{array}}{\mathcal{Y}} \to E$
Or elimination: $\dfrac{\mathcal{X} \vee \mathcal{Y} \quad \begin{array}{|c|}\hline [\mathcal{X}] \\ \vdots \\ \mathcal{Z} \\ \hline\end{array} \quad \begin{array}{|c|}\hline [\mathcal{Y}] \\ \vdots \\ \mathcal{Z} \\ \hline\end{array}}{\mathcal{Z}} \vee E$	Implication introduction: $\dfrac{\begin{array}{|c|}\hline [\mathcal{X}] \\ \vdots \\ \mathcal{Y} \\ \hline\end{array}}{\mathcal{X} \to \mathcal{Y}} \to I$
Or introduction: $\dfrac{\mathcal{X}}{\mathcal{X} \vee \mathcal{Y}} \vee I$	Universal Introduction: $\dfrac{\phi[t/x]}{\forall x.\phi} \forall I$
False elimination: $\dfrac{\boldsymbol{F}}{\mathcal{X}} \boldsymbol{F}E$	Universal Elimination: $\dfrac{\forall x.\phi}{\phi[t/x]} \forall E$
Negation elimination: $\dfrac{\begin{array}{c}\mathcal{X} \\ \neg\mathcal{X}\end{array}}{\boldsymbol{F}} \neg E$	Existential Introduction: $\dfrac{\phi[t/x]}{\exists x.\phi} \exists I$
Negation Introduction: $\dfrac{\begin{array}{|c|}\hline [\mathcal{X}] \\ \vdots \\ \boldsymbol{F} \\ \hline\end{array}}{\neg\mathcal{X}} \neg I$	Existential Elimination: $\dfrac{\exists x.\phi \quad \begin{array}{|c|}\hline [\phi[t/x]] \\ \vdots \\ \psi \\ \hline\end{array}}{\psi} \exists E$
Excluded Middle: $\overline{\mathcal{X} \vee \neg\mathcal{X}}$	

Table 1: Natural Deduction Rules

2. (a) Prove the following statement.
For all integers a, b and c, the greatest common divisor $\gcd(a, bc)$ divides the product $\gcd(a, b) \times \gcd(a, c)$. [4 marks]

(b) Let a and b be integers such that $a = 1 \pmod 2$ and $b = 3 \pmod 4$. Show that $a^2 + b$ divides 4. [5 marks]

(c) In Peano Arithmetic, using first-order logic define axioms for:

(i) Subtraction over the natural numbers, $x - y$, where x and y are natural numbers.

(ii) Exponentiation over natural numbers, forming b^e, where b and e are both natural numbers, b is the base and e is the exponent.

[5 marks]

(d) The following GNU Prolog program models a road system, with no loops. The program objective is to print out all the destinations available from a queried starting point indicating whether they are busy or not. Facts in this scenario are represented with: a predicate `road(From,To)` indicating that there is a road from a location `From` to a location `To`; and, a predicate `busy(Location)` indicating that the destination `Location` is busy.

```
road(a,b).
road(a,d).
road(b,c).
busy(c).

route(A,B):- road(A,B).
route(A,B):- road(A,C),route(C,B).

result(check(Goal),free):- check(Goal),!.
result(check(Goal),busy).

check(Goal):- busy(Goal),!,fail.
check(Goal).

test(Start,Goal,Result):-
      route(Start,Goal),result(check(Goal),Result).

printAll(Start):- test(Start,Goal,Result),
       write(Goal),write(':'),write(Result),nl,fail.
```

Briefly explain the output produced with a query `?-printAll(a).` In addition, with the help of an execution tree, analyse how Prolog resolves a query `?-test(a,c,Result).` In your answer, indicate and discuss the presence or absence of any backtracking steps. [6 marks]

[Total marks 20]

3. (a) **Advisory Word Limit: 250 Words in total for (a)**
Discuss what is meant by undecidability of First Order Logic (FOL), and in your own words explain how this can be demonstrated. [4 marks]

(b) Consider the following statements as axioms:

(A1) Every food item belongs to a food group

(A2) If a food item belongs to the dairy food group or is sugary, it does not belong to the vegetables food group

(A3) Every food item that can be cooked somewhere can be eaten safely

(A4) There exists some food item that can be cooked in the oven

(A5) Onions do not belong to the dairy food group but can be cooked in the oven

Assume that we represent these axioms with the following first order logic statements:

(A1) $\forall x.(\exists y.B(x,y))$

(A2) $\forall x.((B(x,dairy) \vee S(x)) \rightarrow \neg B(x,vegetables))$

(A3) $\forall x.((\exists y.C(x,y)) \rightarrow E(x))$

(A4) $\exists x.C(x,oven)$

(A5) $(\neg B(onions,dairy)) \wedge C(onions,oven)$

Use first order logic resolution to show that the axioms imply that "If onions belong to the vegetables food group, then onions are not sugary and can be eaten safely". [8 marks]

(c) Consider the set of strings S composed by 0s and 1s defined as follows:

- Base step: the empty string ϵ is in the set S (i.e., $\epsilon \in S$).
- Recursive step: given a string λ in the set S (i.e., $\lambda \in S$), and a character a that can be 0 or 1 (i.e., $a \in \{0,1\}$), the string $a \cdot \lambda$ is in the set S (i.e., $a \cdot \lambda \in S$).

Let $num_1(\lambda)$ be a function indicating the number of 1s in the string $\lambda \in S$, e.g., $num_1(1011) = 3$. We can define this function as follows:

- Base step: $num_1(\epsilon) = 0$
- Recursive step: given a character $a \in \{0,1\}$, and a string $\lambda \in S$, $num_1(a \cdot \lambda) = \ell_a + num_1(\lambda)$ where $\ell_a = 1$ if a is equal to 1 and $\ell_a = 0$ otherwise

This function has a further property with respect to concatenation: given two strings $\lambda_1, \lambda_2 \in S$, $num_1(concat(\lambda_1, \lambda_2)) = num_1(\lambda_1) + num_1(\lambda_2)$.

Using definitions and properties of *concat*, *reverse* and num_1, prove that for any string λ: $num_1(concat(\lambda, reverse(\lambda)))$ is an **even number**.

You may consider zero to be an even number for this proof. You may use string properties and definitions listed in Table 2 without further justification or proof, but you must note which rule/result you are using. Basic properties of natural

numbers can also be used with appropriate attribution but without additional proofs (e.g., commutativity of addition, etc). If you use any additional property beside those provided, please provide its proof or justification. [8 marks]

[Total marks 20]

Table 2: Properties and Definitions of Strings

Property Name	**Property**
Elements	• a is a character $\in \{0, 1\}$ • λ_i is a string • ϵ is the empty string • $a \cdot \lambda$ is the concatenation between a character and a string
Identity ϵ for $\cdot$	$a \cdot \epsilon = \epsilon \cdot a$
Definition of *reverse*	• Base step: $reverse(\epsilon) = \epsilon$ • Recursive step: $reverse(a \cdot \lambda) = concat(reverse(\lambda), (a \cdot \epsilon))$
Definition of *concat*, left hand side	• Base step: $concat(\epsilon, \lambda_1) = \lambda_1$ • Recursive step: $concat(a \cdot \lambda, \lambda_1) = a \cdot concat(\lambda, \lambda_1)$
Definition of *concat*, right hand side	• Base step: $concat(\lambda_1, \epsilon) = \lambda_1$ • Recursive step: $concat(\lambda_1, \lambda \cdot a) = concat(\lambda_1, \lambda) \cdot a$
num_1 on *concat*	$num_1(concat(\lambda_1, \lambda_2)) = num_1(\lambda_1) + num_1(\lambda_2)$

***** END OF PAPER *****