







## Assignment 1

### Logic-based KR & Logic Programming

#### 1. Assignment Aims

- To learn how to represent knowledge using propositional and first-order logic.
- To become skilled in reasoning using natural deduction and resolution.
- To gain experience in logic programming using Prolog.

#### 2. Learning Outcomes

- Analyse in depth a variety of established logic-based knowledge representation languages and use them to formulate sound and complete reasoning.
- Apply logic programming techniques to solve a significant problem requiring knowledge representation and reasoning.

#### 3. Assessment Brief

##### Part 1 (40%)

##### **Exercise 1: Propositional Logic Representation [5%]**

Express the following declarative sentences in propositional logic; in each case state what your respective propositional symbols  $p$ ,  $q$ , etc. mean:

- a) [0.5%] The contract is not valid unless it has been signed by all parties and validated by a public authority.
- b) [0.5%] Tomorrow we will either visit the museum or walk down the pier, but not both.
- c) [0.5%] If the parliament approves the no-confidence motion, the government must resign or ask for a snap election.

For each of the following invalid inferences, give examples of declarative sentences in natural language for the atoms  $p$  and  $q$ , such that the premises are true, but the conclusion false.

- d) [0.5%]  $p \vee q \vdash p \wedge q$
- e) [0.5%]  $p \rightarrow q \vdash p \vee q$

##### **Exercise 2: Conjunctive Normal Form [5%]**

Construct formula  $\phi$  in Conjunctive Normal Form based on the following truth table. Make the formula as simple as possible using known equivalences of propositional logic.

p	q	r	$\phi$
T	T	T	F
T	T	F	T
T	F	T	F
F	T	T	F
T	F	F	T
F	T	F	F
F	F	T	F
F	F	F	T

### Exercise 3: Natural Deduction for Propositional Logic [5%]

Use natural deduction to prove the validity of the following inference:

$$(p \wedge q) \vee (p \wedge r) \vdash p \wedge (q \vee r)$$

### Exercise 4: First-Order Logic Representation [10%]

Consider an interpretation  $\langle D, I \rangle$ , where D is the set of natural numbers and I is a mapping as follows:

$$I(a) = 0$$

$$I(b) = 1$$

$$I(c) = 3$$

$$I(f): n \rightarrow n^2$$

$$I(g): (m, n) \rightarrow m + n$$

$$I(P) = \text{"is an even number"}$$

$$I(Q) = \{(m, n) \mid m \text{ divides } n\}$$

Decide if the following formulas are true or false under interpretation  $\langle D, I \rangle$ , showing the full decision proof:

- a) [2%]  $P(f(b))$
- b) [2%]  $P(g(a, b))$
- c) [2%]  $Q(c, b)$
- d) [2%]  $\forall x Q(x, x)$
- e) [2%]  $\exists x Q(x, f(x))$

### Exercise 5: Resolution [15%]

- a) [10%] Use (standard) resolution to determine whether the following set of premises entails  $Q$ . In your answer you should show the resolution tree.

$$\{P \rightarrow \neg R \wedge V, \quad P \vee Q, \quad \neg R \rightarrow \neg U, \quad V \rightarrow U\}$$

- b) [5%] Is it possible to apply SLD resolution for the same problem? Justify your answer.

## **Part 2 (60%)**

For this part you will implement the well-known game of Blackjack (or Twenty-One) in Prolog. All source files need to be submitted in a separate zipped file.

We consider a simplified version of Blackjack, where there is only one (human) player and an AI dealer, playing without bets. In each round, the winner is the player whose cards (their *hand*) sum up to the highest amount of points, without exceeding 21. Dealer or player *bust* when their hand exceeds 21.

In Brightspace, you can find a partial implementation to get you started. You are free to modify it in any way you see fit, or even to create your own implementation from scratch.

### **Exercise 6: Representing the Deck [5%]**

Blackjack is played with a standard 52-card deck, containing four suits (diamonds, clubs, hearts and spades), with each suit containing thirteen ranks (ace, 2 through 10, jack, queen and king). Write Prolog code (deck.pl) that implements the following:

- a) [3%] Represent possible suits, ranks and cards.
- b) [2%] Create a standard 52-card deck.

### **Exercise 7: Dealing Hands [10%]**

In a Blackjack round, two cards are dealt to each player from the top of the deck in an alternating manner as follows: one card to the player, one card to the dealer, second card to the player, second card to the dealer. All cards dealt collectively constitute the *table* of each round. The player is then asked if they want to *hit* (get another card) or *stand* (accept no more cards). If they hit, a new card is dealt from the top of the deck. Based on these, write Prolog code (deck.pl) that implements the following:

- a) [3%] Given a deck (as created in Exercise 6b), deal initial cards to player and dealer, resulting in an updated deck and the table (dealt cards).
- b) [3%] Ask player if they want another card (print message on screen), resulting in the player's answer (yes or no).
- c) [4%] Given a deck and a table, deal a card to the player, resulting in updated deck and table.

### **Exercise 8: Printing the Table [10%]**

When cards are initially dealt, the dealer's first card is hidden. After the player has decided whether they want a third card, the hidden card is revealed. Write Prolog code (blackjack.pl) that implements the following:

- a) [6%] Given a hand, print it to the screen.
- b) [4%] Given a table and a game state (initial or otherwise), print the table to the screen.

### Exercise 9: Calculating Points and Scores [20%]

Each card has a value based on its rank. Ranks 2 through 10 get the same amount of points as their rank. Jacks, queens and kings all get 10 points. Aces may either get 1 or 11 points, depending on the player's choice. Based on these, write Prolog code (deck.pl and blackjack.pl) that implements the following:

- a) [3%] Given a card, calculate its points.
- b) [3%] Given a hand, calculate its total points.
- c) [4%] Given a hand (that may contain an ace), calculate its maximum points, without exceeding 21 if possible. For instance: for hand  $\spadesuit Q \heartsuit 3 \clubsuit A$ , the result is 14; for hand  $\spadesuit Q \heartsuit 3 \clubsuit A \spadesuit K$ , the result is 24.
- d) [5%] Given a table, determine the status of the dealer and player (bust, 21, or less than 21).
- e) [5%] Given a table, decide who scores a win. Using retract and assertz, update scores and print them to the screen. Scores are calculated as follows:
  - Player's score increases by 1 if they have 21 and the dealer busts or has less than 21.
  - Dealer's score increases by 1 if they do not bust and their hand's points exceed the player's.
  - Both dealer's and player's score increases by 1 if neither busts and their hands have the same points.
  - In all other cases, scores stay the same.

### Exercise 10: Putting It All Together [15%]

A complete round of Blackjack consists of the following steps:

Step 1: a deck is generated and shuffled.

Step 2: Cards are dealt to players and the initial table is printed to the screen.

Step 3: Scores are calculated and the status of the dealer and player is determined.

Step 4: if the player has not busted and has less than 21, they are asked if they want another card (which is dealt to them if they say yes).

Step 5: Repeat steps 2 to 4 until the player busts or refuses to get more cards.

Step 6: If the dealer has not busted and has less than 21, the dealer's AI determines if they will get dealt another card. The new table is printed to the screen.

Step 7: Dealer and player scores are updated and printed to the screen.

Write Prolog code (blackjack.pl) that runs a pre-specified number of rounds. You should combine the code you have written in exercises 6 through 9, as well as code that is provided in deck.pl and dealerStrategies.pl that implements shuffling and dealer AI, respectively.

#### 4. **Marking Scheme**

In the assessment brief, a specific number of points is associated with each exercise, which are allocated as follows:

##### Exercise 1

- a) 1 point for a propositional sentence that is logically equivalent to the natural language one. 0.5 points when only part of the sentence is correctly represented. 0 points if the provided propositional sentence is in no way related to the natural language sentence.
- b) Same as (a).
- c) Same as (a).
- d) 0.5 point for a valid natural language example for p and 0.5 point for a valid natural language example for q.
- e) Same as (d).

##### Exercise 2

1 point for the correct encoding of each True line of the truth table (3 total). 2 point for successfully simplifying the original formula using known equivalences of propositional logic.

##### Exercise 3

1 point for a correctly structured derivation, 1 point for a correctly structured sub-derivation, 2 points for a successful sub-derivation and 1 point for a successful derivation.

##### Exercise 4

For all subquestions, 0.5 for a correct decision (true or false), 1.5 for a logically correct decision proof.

##### Exercise 5

1 point for the correct translation of each sentence (premises and conclusion) into a resolution clause (5 total). A further 5 points for a correct resolution tree.

##### Exercises 6-10

10% of the points allocated to each subquestion is for well-commented code, 70% for code that delivers the complete functionality as stated and the remaining 20% is for code that is not only functionally correct but is as compact as possible, making good use of recursion and modularisation.

#### 5. **Grading Rubric**

A detailed grading rubric is available on the next page.



Marks Available	Part 1 – Solutions to logic-based exercises that show...	Part 2 – Prolog code that is...
0 to 29%	only ability to deal with basic facts and concepts, at a level below the learning outcome threshold	barely executable or correct and/or only addresses a significantly small subset of the required functionality
30-39%	knowledge of concepts that falls short of prescribed range and does not address the exercise tasks	executable or correct under restricted conditions and/or only address a small subset of the required functionality
40-49%	knowledge of concepts within prescribed range but fails to adequately address the exercise tasks	executable or correct under some conditions and/or addresses a reasonable subset of the required functionality
50 - 59%	systematic understanding of knowledge that leads however to only partially addressing the exercise tasks	executable and correct under most circumstances and addresses a significant subset of the required functionality
60 - 69%	nearly excellent in knowledge and addressing the most significant aspects of the exercise tasks	executable and correct under almost all circumstances and addresses almost all of the required functionality, but is unnecessarily large, showing lack of understanding of recursion or modularisation.
70 - 79%	excellent demonstration of knowledge, addressing all significant aspects of the exercise tasks	executable and correct under all circumstances and addresses all required functionality while also showing evidence of use of recursion and modularisation
80-89%	striking and insightful demonstration of knowledge, addressing almost all aspects of the exercise tasks	executable and correct under all circumstances and addresses all required functionality while also showing strong evidence of use of recursion and modularisation leading to a compact result.
≥ 90%	outstanding and insightful demonstration of knowledge and perfect or near-perfect addressing of all aspects of the exercise tasks	executable and correct under all circumstances and shows outstanding and insightful programming skills that deliver all required functionality with a minimal code size.