Concordia University

Department of Computer Science and Software Engineering

SOEN 331-S:

Formal Methods for Software Engineering

Assignment 1 on logic, discrete structures and construction techniques

Dr. Constantinos Constantinides, P.Eng.

`constantinos.constantinides@concordia.ca`

February 15, 2024

General information

Date posted: Thursday, 15 February, 2024.

Date due: Thursday, 7 March, 2024, by 23:59.

Weight: 10% of the overall grade.

Introduction

You should find one partner and between the two of you should designate a team leader who will submit the assignment electronically. There are seven (**7**) problems in this assignment. The total weight of the assignment is **100** points.

Ground rules

1. This is an assessment exercise. You may not seek any assistance while expecting to receive credit. **You must work strictly within your team and seek no assistance for this assignment (e.g. from the teaching assistants, fellow classmates and other teams or external help)**. You should **not** discuss the assignment during tutorials. I am available to discuss clarifications in case you need any.

2. **Both partners are expected to work relatively equally on each problem**. Accommodating a partner who did not contribute will result in a penalty to **both**. You cannot give a "free pass" to your partner, with the promise that they will make up by putting more effort in a later assignment.

3. **You must prepare this assignment in LaTeX**, by using a compiler on your local machine (see Resources on course website). Submissions not prepared using LaTeXwill **not** be accepted.

4. If there is any problem in the team (such as lack of contribution, etc.), you must contact me as soon as the problem appears.

5. Make sure you have an up-to-date VPN client on your local machine and you are able to access our electronic submissions system.

6. Late submissions will **not** be accepted, and no submissions will be accepted by email.

PROBLEM 1: Propositional logic (5 pts)

You are shown a set of four cards placed on a table, each of which has a **number** on one side and a **color** on the other side. The visible faces of the cards show the numbers **9** and **11**, and the colors **blue**, and **yellow**.

Which card(s) must you turn over in order to test the truth of the proposition that *"If the face of a card is* ***blue****, then it has a* ***prime*** *number on the other side"*? Explain your reasoning by deciding for each card whether or not it should be turned over and why.

PROBLEM 2: Predicate logic (14 pts)

Part 1 (8 pts)

Consider types *Object*, and the binary relation *Orbits* over the domain of all celestial objects, which is codified by clause `object/1` in Prolog (available in `solar.pl`):

1. (2 pts) Construct a formula in predicate logic to define a planet, where planet is defined as an object whose mass is greater than $0.33 \times 10^{24} KG$, and which it orbits around the sun. For all practical purposes, you may ignore the $10^{24} KG$ factor.

   Use the formula for *Planet* to construct a formula that defines the binary relation *is_satellite_of* in terms of the binary relation *Orbits*. A satellite is an object that orbits around a planet.

   For Questions 2 and 3, you must a) include your interaction with the Prolog interpreter in your main submission, and b) extend `solar.pl` with rules so that we can run and assess your code.

2. (3 pts) (**PROGRAMMING**) Map your formulas to Prolog rules `is_planet/1`, and `is_satellite_of/2`, and demonstrate how it works by executing both ground- and non-ground queries. **Identify the type of each query**. Example queries (type not shown) are shown below:

```
?- is_planet(pluto).
false


?- is_planet(mars).
true


?- is_planet(P).
P = mercury;
P = venus;
```

```
P = earth;
P = mars;
P = jupiter;
P = saturn;
P = uranus;
P = neptune;
false

?is_satellite_of(moon, Planet).
Planet = earth;
false

is_satellite_of(S, mars).
S = deimos;
S = phobos;
false
```

3. (3 pts) (<u>**PROGRAMMING**</u>) Construct a Prolog rule `obtain_all_satellites/2` that succeeds by returning a collection of all satellites of a given planet. Example:

```
?- obtain_all_satellites(earth, L).
L = [moon]

?- obtain_all_satellites(mars, L).
L = [deimos, phobos]

?- obtain_all_satellites(venus, L).
[]

?- obtain_all_satellites(saturn, L).
L = [atlas, calypso, helene]
```

Part 2: Categorical propositions (2 pts)

In the domain of all integers, let $number(x)$ denote the statement "x is a number", and $composite(x)$ denote the statement "x is a composite." Formalize the following sentences and indicate their corresponding formal type.

1. "Some numbers are not composite."
2. "No numbers are prime."
3. "Some numbers are not prime."
4. "All numbers are prime."

Part 3: Categorical propositions (4 pts)

For subject S and predicate P, we can express the Type A categorical proposition as

$$\forall\, s : S \mid s \in P$$

1. Prove formally that negating A is logically equivalent to obtaining O (and vice versa).
2. Prove formally that negating E is logically equivalent to obtaining I (and vice versa).

PROBLEM 3: Temporal logic (22 pts)

Part 1 (16 pts)

The behavior of a program is expressed by the following temporal formula:

$$\Box \begin{bmatrix} \textbf{start} \rightarrow (\neg\phi \vee \neg\psi) \\ \\ \textbf{start} \rightarrow \chi \oplus \tau \\ \\ \phi \vee \chi \rightarrow \bigcirc(\pi \mathcal{R} \omega) \\ \\ x \wedge \bigcirc\omega \rightarrow \bigcirc^3 \pi \\ \\ \tau \rightarrow \bigcirc^2(\tau \mathcal{W} \rho) \\ \\ \omega \wedge \bigcirc\rho \rightarrow \bigcirc^2 \rho \\ \\ \psi \wedge \tau \rightarrow \bigcirc(\rho \mathcal{U} \tau) \\ \\ \rho \wedge \bigcirc\tau \rightarrow \bigcirc^2 \rho \\ \\ \phi \wedge \tau \rightarrow \bigcirc(\tau \mathcal{R} \rho) \end{bmatrix}$$

1. (9 pts) Visualize all models of behavior.
2. (7 pts) Make observations on the visualization by specifying exact conditions about termination, non-termination and consistency (or lack thereof), if any exist.

Part 2 (6 pts)

1. (2 pts) Formalize and visualize the following requirement: "If none of ϕ or ψ are invariants, then starting from time $= i + 2$, χ will eventually become true and it will remain true up to and including the moment τ first becomes true. Note that there exists no guarantee that τ ever becomes true."

2. (2 pts) Describe the visualize the following requirement: $(\neg\alpha \vee \neg\beta) \rightarrow \bigcirc\Diamond(\gamma\ \mathcal{U}\ \delta)$

3. (2 pts) Describe the visualize the following requirement:

$$(\bigcirc\tau \wedge \bigcirc\Diamond\Box\chi) \rightarrow \bigcirc^2(\phi\ \mathcal{W}\ \psi)$$

PROBLEM 4: Unordered structures (14 pts)

Consider the following set:

$$\textit{Languages} = \{\textit{Ruby}, \textit{Go}, \textit{Lisp}, \textit{Rust}, C, \textit{Groovy}, \textit{Python}, \textit{Clojure}, \{\textit{Lua}, \textit{Groovy}, C\}\}$$

Answer the following questions:

1. (1 pt) Provide a description (in plain English) on what is meant by $\mathcal{P}\textit{Languages}$.
2. (1 pt) Describe in detail (a) what the expression $\textit{Favorites} : \mathcal{P}\textit{Languages}$ signifies and (b) how it should be interpreted. (c) What could be any legitimate value for variable *Favorites*?
3. (1 pt) What does the expression $\textit{Favorites} = \mathcal{P}\textit{Languages}$ signify and (b) Is semantically equivalent to $\textit{Favorites} : \mathcal{P}\textit{Languages}$?
4. (1 pt) Is $\{\textit{Lua}, \textit{Groovy}, C\} \in \mathcal{P}\textit{Languages}$? Explain in detail.
5. (1 pt) Is $\{\{\textit{Lua}, \textit{Groovy}, C\}\} \subset \mathcal{P}\textit{Languages}$? Explain in detail.
6. (1 pt) What is the difference between the following two variable declarations and are the two variables atomic or non-atomic?
   (a) λ_1: Languages
   (b) λ_2: $\mathcal{P}\textit{Languages}$
7. (1 pt) In the expression $\textit{Library} = \{C, \textit{Ruby}, \textit{Go}\}$, where the variable is used to hold any collection that can be constructed from *Languages*, what is the *type* of the variable?
8. (1 pt) Is $\{\varnothing\} \in \mathcal{P}\textit{Languages}$? Explain.
9. (6 pts) (**<u>PROGRAMMING</u>**) Use Common LISP to implement the functions below and demonstrate the behavior of each function with a listing of your interaction with the language environment (to be included in the main submission). **Do not use imperative constructs.**

is-memberp Returns true if its first argument is found to be a member in the set provided by the second argument. Returns nil (false) otherwise.

equal-setsp Returns true if its two arguments are two identical sets. Returns nil (false) otherwise. You may assume that neither argument (set) contains any redundancies.

PROBLEM 5: Ordered structures (15 pts)

Consider the Queue Abstract Data Type (ADT), $\mathbb{Q}$, defined over some generic type $\mathbb{T}$ and defined using two stacks, Σ_1 and Σ_2, where a Stack ADT is implemented as a list, Λ. (*Hint*: This description implies that the only available operations are those defined for a list.)

1. (10 pts) Define operations $Enqueue(\mathbb{Q}, \mathbb{T})$ and $Dequeue(\mathbb{Q})$.

2. (3 pts) (**PROGRAMMING**) Use Common LISP to define two stacks (`stack1` and `stack2`) as global variables to hold the collection, and implement functions `enqueue` and `dequeue`. Place your code in file `queue-adt.lisp`, and include your interaction with the language environment to demonstrate the behavior of your code.

3. (2 pts) (**PROGRAMMING**) Implement operations *head*, *tail* and *cons* in Prolog and demonstrate their usage (include your interaction with the language environment in your submission). Consider the following example executions:

```
?- head([a, b, c, d], H).
H = a .
?- head([x], H).
H = x .
?- head([], H).
false.
?- tail([a, b, c, d], T).
T = [b, c, d] .
?- tail([x], T).
T = [] .
?- tail([], T).
false.
?- cons(a, [b, c], NewList).
NewList = [a, b, c] .
?- cons(a, [], NewList).
```

```
NewList = [a] .

?- cons([], [], NewList).

NewList = [[]] .
```

PROBLEM 6: Binary relations, functions and orderings (20 pts)

Part 1 (6 pts)

1. (2 pts) Consider the binary relation R: "is of type" in the domain of types in the Java API. Prove that R is a *partial order*.

2. (2 pts) Given the set of vertices

$$\begin{aligned} V_1 =\{&\mathit{Dictionary}, \mathit{TreeMap}, \mathit{SortedMap}, \mathit{AbstractMap}, \mathit{Hashtable}, \\ &\mathit{LinkedHashMap}, \mathit{Map}, \mathit{NavigableMap}, \mathit{HashMap}\} \end{aligned}$$

   and the set of edges

$$\begin{aligned} E =\{&(\mathit{LinkedHashMap}, \mathit{HashMap}), (\mathit{HashMap}, \mathit{AbstractMap}), (\mathit{HastTable}, \mathit{Dictionary}), \\ &(\mathit{TreeMap}, \mathit{AbstractMap}), (\mathit{TreeMap}, \mathit{NavigableMap}), (\mathit{NavigableMap}, \mathit{SortedMap}), \\ &(\mathit{Hashtable}, \mathit{Map}), (\mathit{AbstractMap}, \mathit{Map}), (\mathit{SortedMap}, \mathit{Map})\} \end{aligned}$$

   prove that (V_1, R) is a *poset*.

3. (2 pts) Create a Hasse Diagram to visualize V, R, and identify *maximal* and *minimal* elements.

Part 2 (6 pts)

1. (2 pts) Consider the binary relation $\subseteq$: "is subset of" in the domain of sets. Prove that $\subseteq$ is a partial order.

2. (2 pts) Given the set $V_2 = \{a, b, c\}$, prove that $(P(V_2), \subseteq)$ is a poset.

3. (2 pts) Create a Hasse Diagram to visualize $(P(V_2), \subseteq)$, and identify *maximal* and *minimal* elements.

Part 3 (8 pts)

Consider (V_1, R) and $(P(V_2), \subseteq)$ from parts 1 and 2 respectively. Assume the following mapping, captured by variable *map*:

$$\begin{aligned} map = \{&\mathit{Dictionary} \mapsto \{a, b, c\}, \\ &\mathit{AbstractMap} \mapsto \{a, b, c\}, \\ &\mathit{NavigableMap} \mapsto \{c\}, \\ &\mathit{TreeMap} \mapsto \varnothing\} \end{aligned}$$

Is *map* a function? Discuss **in detail** (in plain english and formally) all applicable properties (total vs. partial function, injectivity, surjectivity, bijection, order preserving, order reflecting, order embedding, isomorphism).

PROBLEM 7: Construction techniques (10 pts)

Define a function $\mathit{compress} : \mathit{lists}(\mathbb{T}) \to \mathit{lists}(\mathbb{T})$ that accepts a list argument (of some generic type $\mathbb{T}$) and returns a list without any *subsequent* redundancies, e.g.

$$\mathit{compress}(\langle a, a, b, b, b, c, c, a, b\rangle) = \langle a, b, c, a, b\rangle$$

1. (2 pts) Transform the definition into a computable function.

2. (3 pts) Define f recursively.

3. (2 pts) Unfold your definition for $\mathit{compress}(\langle a, a, b, b, c, a\rangle)$.

4. (3 pts) (**PROGRAMMING**) Map your definition into a Common LISP function. Place your code in file `compress.lisp`.

   Your code should behave as follows:

```
> (compress '(a a a b c d d e))
(A B C D E)
> (compress '())
NIL
> (compress '(a b c c c a))
(A B C A)
> (compress '(a a b b c c c c a b b))
(A B C A B)
```

What to submit

Your main submission consists of a `.tex` file and its corresponding `.pdf` compilation. All code should (1) be embedded in your main submission file and (2) be included as supporting documentation in separate files, and placed in a folder called `/code`. All demonstrations of interactions with a language environment should be embedded in your main submission file.

Place all the above files (`.tex`, `.pdf`, and the `/code` folder) into a folder that is named after you and your partner, where the name of the person to submit goes first, e.g. if Lisa Gerrard and Nick Cave were partners and Lisa were the one to submit, then the folder is called `/gerrard-cave`. Zip your folder and submit it at the Electronic Assignment Submission portal at

(`https://fis.encs.concordia.ca/eas`)

under **Assignment 1**.

END OF ASSIGNMENT.