SOEN 331 - S: Formal Methods for Software Engineering

Assignment 1

Nathan Grenier, Nirav Patel

March 6, 2024

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.

Solution:

We're trying to prove the proposition:

$$p \to q$$

Where:

- p represents the proposition that the card is blue, and
- q represents the proposition that the other side of the card has a prime number on its face.

1. **Blue Card:** You **should** flip this card to prove the proposition. We can apply modus ponens to the proposition to prove it. This would show that if the card is blue, then the other side has a prime number.

$$\begin{array}{c} p \to q \\ p \\ \hline \therefore q \end{array}$$

2. **9 Card:** You **should** flip this card to prove the proposition. We can apply modus tollens to the proposition to prove it. This would show that if the other side of the

card is not a prime number, then the card is not blue.

$$\begin{array}{c} p \to q \\ \neg q \\ \hline \therefore \neg p \end{array}$$

3. **Yellow Card:** You **should not** flip this card to prove the proposition. The proposition in question is only concerned with the blue cards. It makes no assertion about yellow cards. Flipping this card would fall under the category of affirming the consequent, which is not a valid way to prove the proposition.

4. **11 Card:** You **should not** flip this card to prove the proposition. The proposition makes no claim that cards with prime numbers on them must have blue on the other side. This would fall under the category of denying the antecedent which is a logical fallacy.

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 or equal to $0.33 \times 10^{24} KG$, and which it orbits around the sun. For all practical purposes, you may ignore the $10^{24} KG$ factor.

   Solution:

   - Let $M(x)$ be that the mass of object x greater than or equal to $0.33 \times 10^{24} KG$.
   - Let $O(x, y)$ be that x orbits around y.
   - Let $P(x)$ be that x is a planet.

   The formula to define a planet is:

$$P(x) \equiv M(x) \wedge O(x, sun)$$

   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.

   Solution:

   - Let $S(x, y)$ mean that x is a satellite of y.
   - Let $O(x, y)$ be that x orbits around y.
   - Let $P(x)$ be that x is a planet.

$$S(x, y) \equiv O(x, y) \wedge P(y)$$

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. **Solution**: Here are the translated prologue rules:

   (a) **is_planet/1**

   ```
   planet(X) :- orbits(X, sun), mass(X, M), M >= 0.33.
   ```

   (b) **is_satellite_of/2**

   ```
   is_satellite_of(X, Y) :- is_planet(Y), orbits(X, Y).
   ```

   List of example queries:

   - Query `?- is_planet(pluto).`
     - Type: Ground Query
     - Result: `false`
   - Query `?- is_planet(mars).`
     - Type: Ground Query
     - Result: `true`
   - Query `?- is_planet(P).`
     - Type: Non-Ground Query
     - Result:

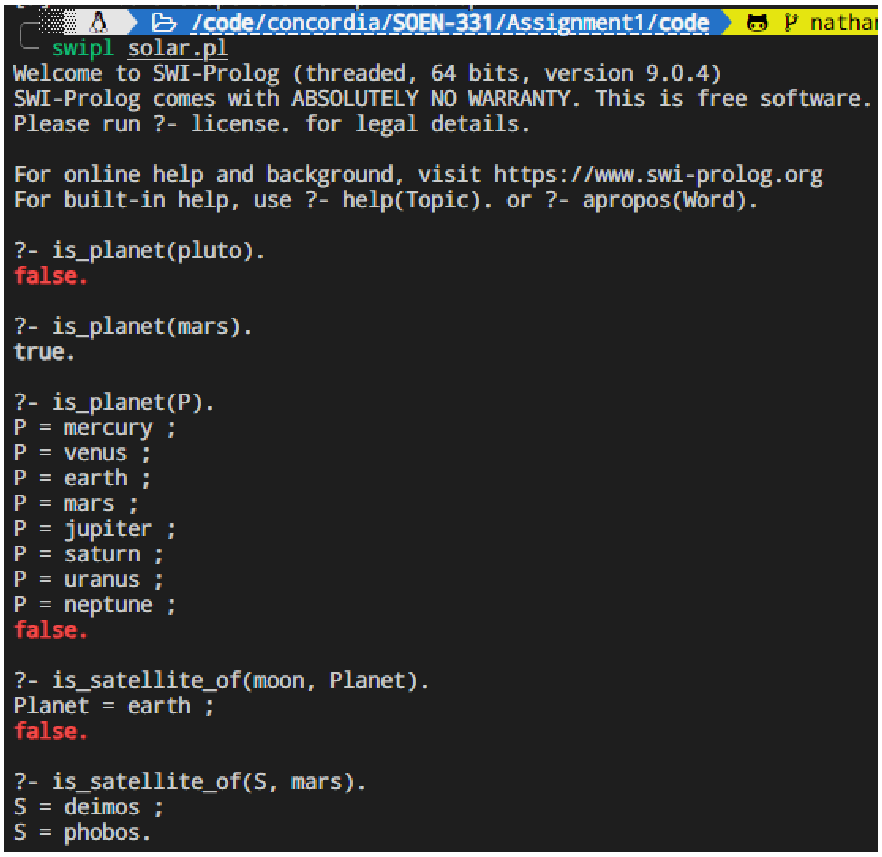
       ```
       P = mercury
       P = venus
       P = earth
       P = mars
       P = jupiter
       P = saturn
       P = uranus
       P = neptune
       false
       ```

- Query `?- is_satellite_of(moon, Planet).`
  - Type: Non-Ground Query
  - Result:

    ```
    Planet = earth
    false
    ```

- Query `?- is_satellite_of(S, mars).`
  - Type: Non-Ground Query
  - Result:

    ```
    S = deimos
    S = phobos
    false
    ```



```
/code/concordia/SOEN-331/Assignment1/code   nathar
└ swipl solar.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- 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.
```

Figure 1: Screenshot of CLI Outputs

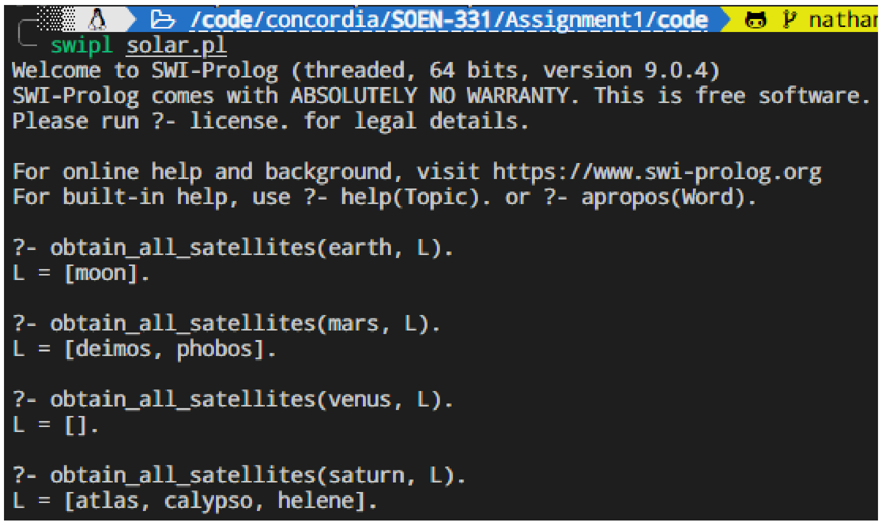
3. (3 pts) (PROGRAMMING) Construct a Prolog rule obtain_all_satellites/2 that succeeds by returning a collection of all satellites of a given planet.

Solution:

obtain_all_satellites/2

```
obtain_all_satellites(Planet, Satellites) :- findall(Satellite
    , is_satellite_of(Satellite, Planet), Satellites).
```

List of example queries:



```
/code/concordia/SOEN-331/Assignment1/code   nathar
swipl solar.pl
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

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

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

?- obtain_all_satellites(venus, L).
L = [].

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

Figure 2: CLI output of the query `?- obtain_all_satellites(earth, Satellites).`

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:

Solution:

1. "Some numbers are not composite."
   - Formalization: $\exists\, x(number(x) \wedge \neg\, composite(x))$
   - Corresponding Formal Type: Particular Negative (O)
2. "No numbers are prime."
   - Formalization: $\forall\, x(number(x) \rightarrow \neg\, prime(x)) \equiv \forall\, x(number(x) \rightarrow composite(x))$
   - Corresponding Formal Type: Universal Affirmative (A)
3. "Some numbers are not prime."
   - Formalization: $\exists\, x(number(x) \wedge \neg\, prime(x)) \equiv \exists\, x(number(x) \wedge composite(x))$
   - Corresponding Formal Type: Particular Affirmative (I)
4. "All numbers are prime."
   - Formalization: $\forall\, x(number(x) \rightarrow prime(x)) \equiv \forall\, x(number(x) \rightarrow \neg\, composite(x))$
   - Corresponding Formal Type: Universal Negative (E)

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).

   Solution:

   - Negation of A: $\neg\, (\forall\, s : S \mid s \in P)$
   - Apply the negation inside the quantifier: $\exists\, s : S \mid s \notin P$
   - This is the Type O categorical proposition (Particular Negative): Some S are not P.
   - We can negate this new expression to obtain the original proposition of type A.

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

   Solution:

   - The categorical proposition of type E is: $\forall\, s : S \mid s \notin P$
   - Negation of E: $\neg\, (\forall\, s : S \mid s \notin P)$
   - Apply the negation inside the quantifier: $\exists\, s : S \mid s \in P$
   - This is the Type I categorical proposition (Particular Affirmative): Some S are in P.
   - We can negate this new expression to obtain the original proposition of type E.

PROBLEM 3: Temporal logic (22 pts)

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

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

Figure 3: Temporal Formula