SOEN 331 (U):Formal Methods for Software Engineering

Assignment 1

Charles Partous (40175854), Mark Ghaby ()

March 7, 2024

Problem 1: Propositional logic

P: card is blue

Q: card is prime

P $\rightarrow$ Q

Card 1: Number 9, Card 2: Number 11, Card 3: Blue , Card 4: Yellow

Card1:

To demonstrate modus tollens, flip the card to prove that a card with a prime number is not blue.

Card2:

No need to flip this card.

No conclusion can be drawn from the outcome because a prime number doesn't imply that a card must be blue.

Card3:

To demonstrate modus ponens, flip this card to prove that blue cards have a prime number.

Card4:

No need to flip this card since a yellow card doesn't necessarily imply a non-prime number.

Problem 2: Propositional logic

Part 1

1. (X orbits Sun AND Mass of X $\geq$ 0,33) $\rightarrow$ X is Planet

   (X is not Planet AND Y is Planet AND X orbits Y) $\rightarrow$ X is satellite of Y

2. is_planet(X) :- orbits(X,sun), mass(X, Mass), Mass $\geq$ 0.33.

   is_satellite_of(X,Y) :- is_planet(Y), orbits(X , Y), not(is_planet(X)).

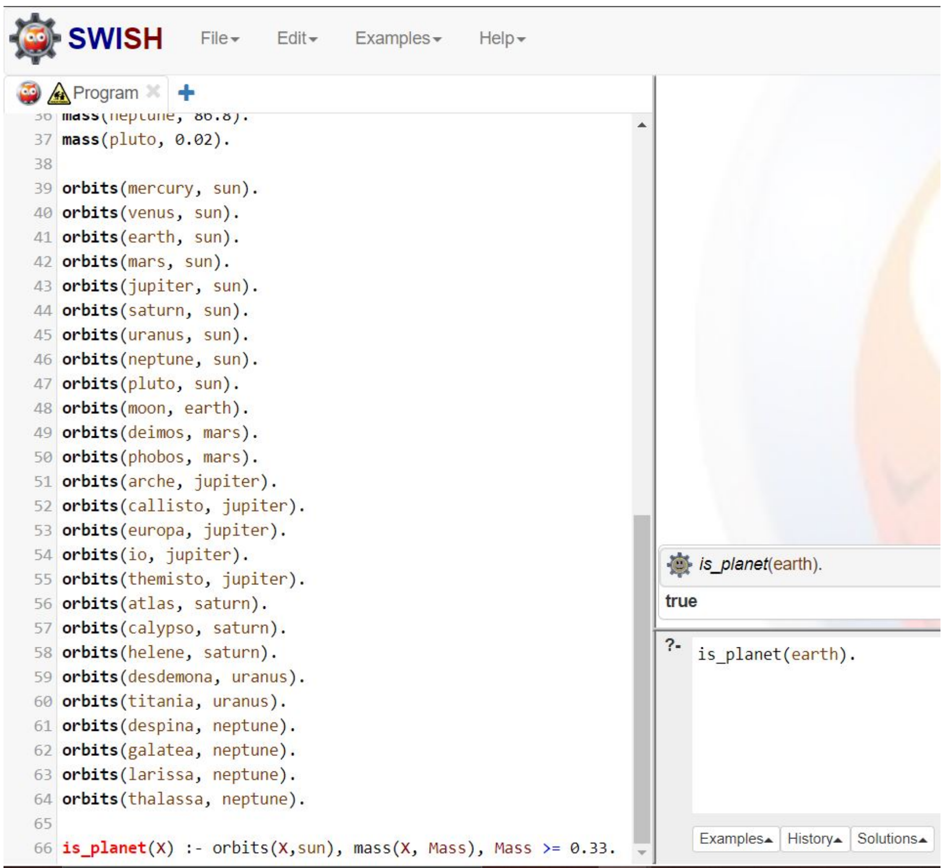

Figure 1: is_planet(earth) - Ground query.

3. obtain_all_satellites(Planet , L) :- findall(X , is_satellite_of(X , Planet) , L).

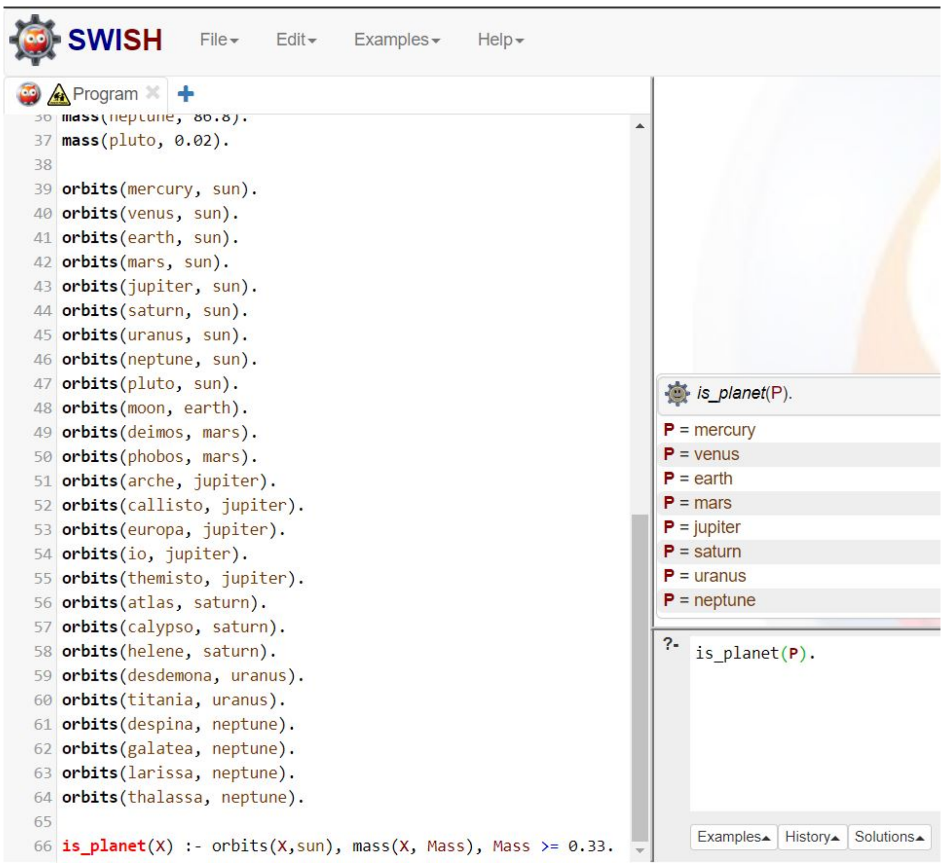

Figure 2: is_planet(P) - Non-ground query.

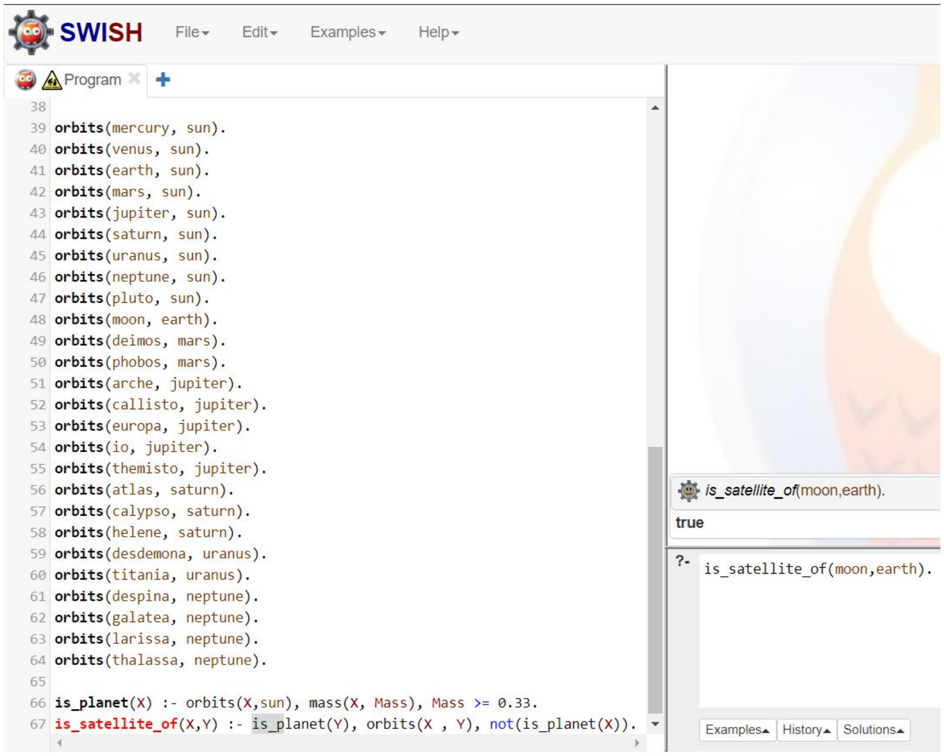

Figure 3: is_satellite_(moon,earth) - Ground query.

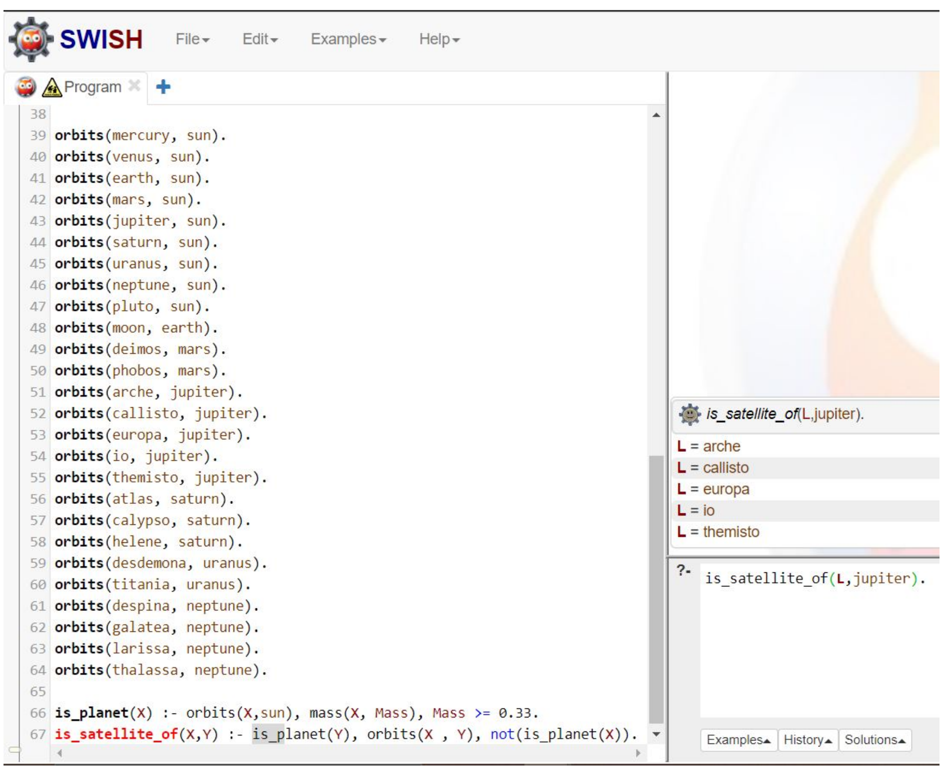

Figure 4: is_satellite_(L,jupiter) - Non-ground query.

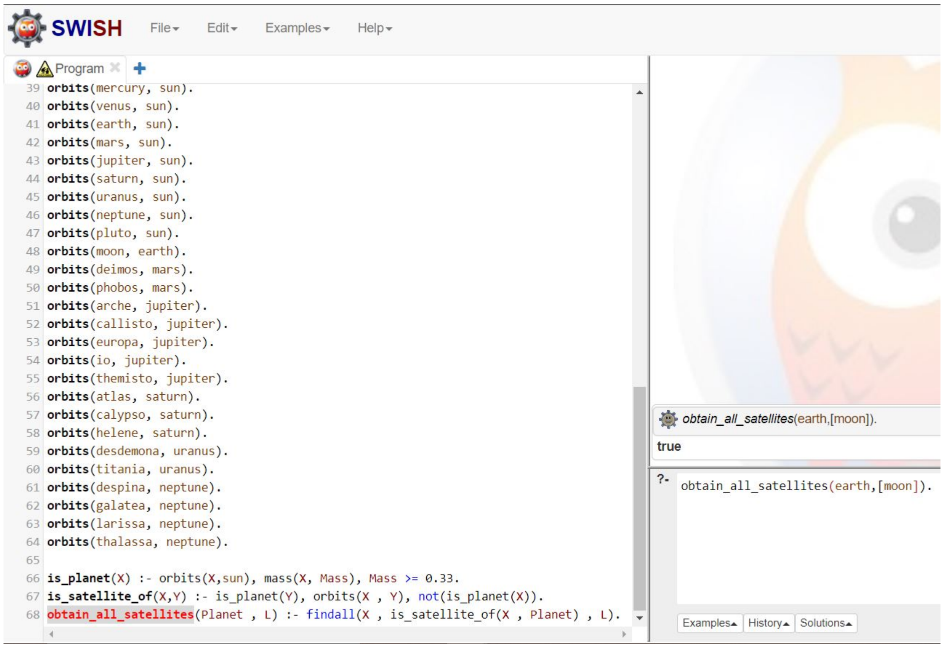

Figure 5: obtain_all_satellites_(earth,[moon]) - Ground query.

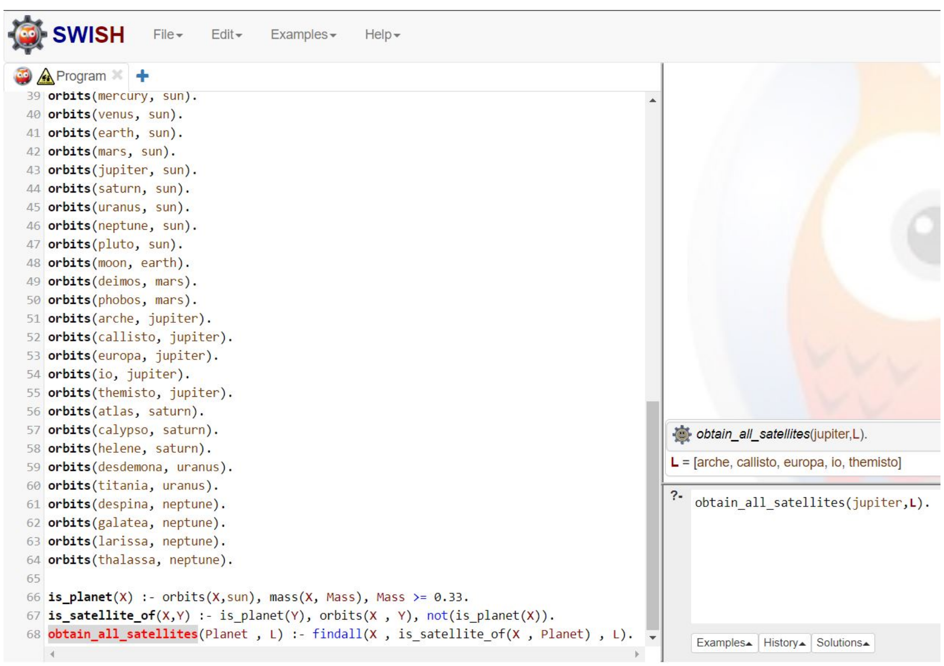

Figure 6: obtain_all_satellites_(jupiter,L) - Non-ground query.

Part 2

1. Type O Particular Negative: $\exists\, x \neg P(x)$
2. Type A Universal Affirmative: $\forall\, x P(x)$
3. Type I Particular Affirmative: $\exists\, x P(x)$
4. Type E Universal Negative: $\forall\, x \neg P(x)$

Part 3

1. The formal definition of Type O categorical propositions is $\exists\, x \neg P(x)$. By negating a type A categorical proposition, and following the laws of logic, we find that:

   $\neg(\forall\, x P(x)) \equiv \exists\, x \neg P(x)$ AND $\neg(\exists\, x \neg P(x)) \equiv \forall\, x P(x)$

2. The formal definition of Type E categorical propositions is $\forall\, x \neg P(x)$. In Negating a type E proposition, the universal quantifier($\forall$) becomes the existential quantifier($\exists$) and the proposition is negated. The result yields the type I proposition ($\exists\, x P(x)$). In the inverse, negating a type I proposition yields a type E proposition.

   $\neg(\forall\, x \neg P(x)) \equiv \exists\, x P(x)$ AND $\neg(\exists\, x P(x)) \equiv \forall\, x \neg P(x)$.

Problem 3: Temporal logic

Part 1

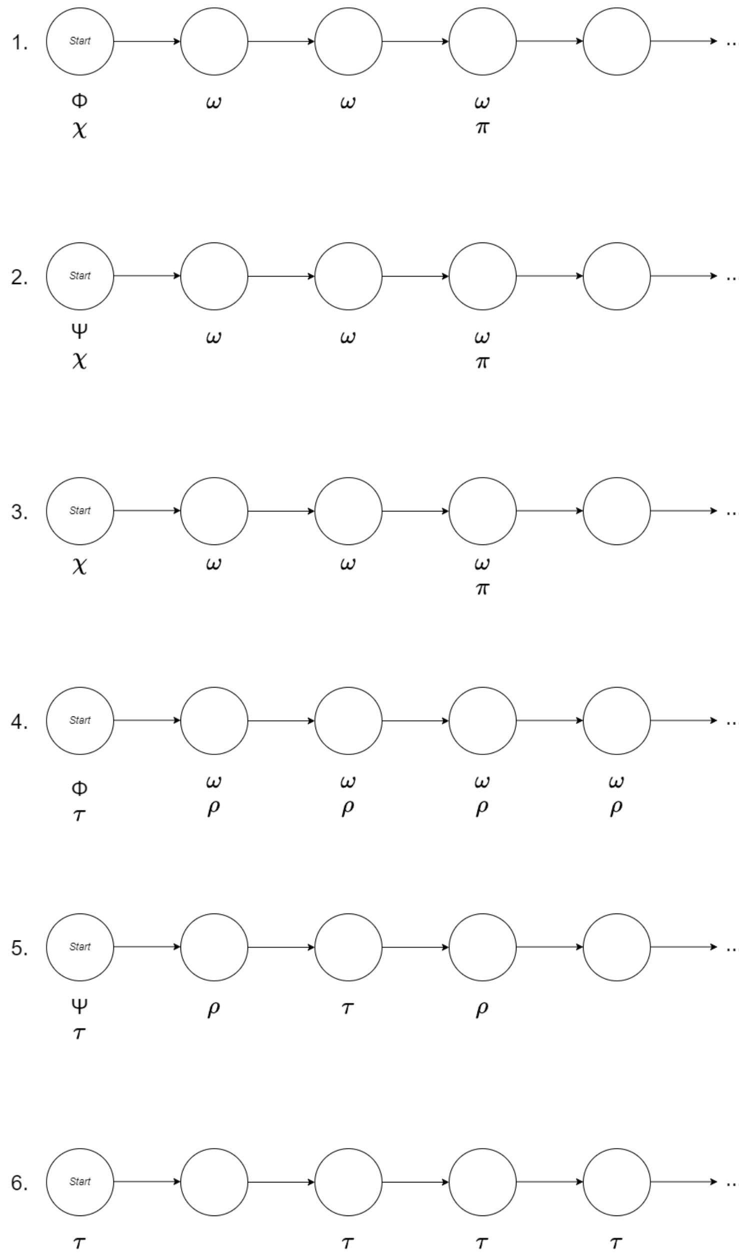

Figure 7: Visualization Problem 3

1.

2. In order for the program to terminate, some conditions need to be met depending on the state of the program. If ω becomes true then for the program to terminate, π must also become true at least once after ω. In the case of τ becoming true then for the program to terminate, ρ must become true after τ. Given any state, the program will terminate eventually.

Part 2

1. $(\neg\Box\phi \wedge \neg\Box\psi) \rightarrow \bigcirc^2(\Diamond(\mathrm{x}\ \mathcal{W}\ \tau))$

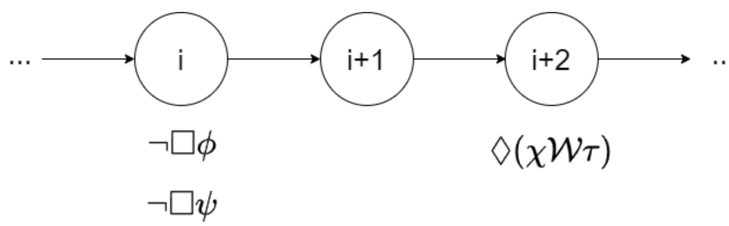

Figure 8: Visualization Problem 3 — Part 2 — 1.

2. If α and β do not hold true at the same time then starting the next state, γ will eventually hold true until the moment δ becomes true. δ is guaranteed to become true.

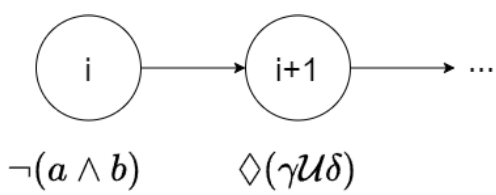

Figure 9: Visualization Problem 3 — Part 2 — 2.

3. If starting at time = i+1, τ becomes true and $\mathcal{X}$ eventually becomes invariant; Then starting in time = i + 2, ϕ becomes true and holds true unless ψ becomes true.

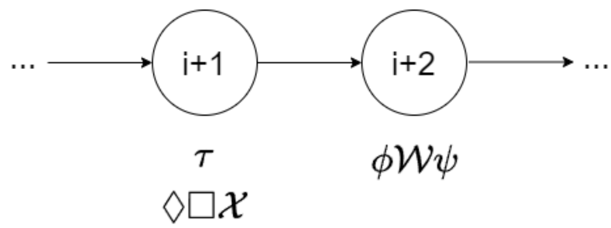

Figure 10: Visualization Problem 3 — Part 2 — 3.

Problem 4: Unordered structures

1. PLanguages would be a set containing all possible subsets of Languages, namely the powerset of languages

2. (a) Favorites is a non-atomic element, or a subset, of the powerset of the Languages set.

   (b) It should be interpreted as a set

   (c) Favorites can be the empty set, any subset of the power set of Languages, or Languages itself.

3. (a) Favorites is equal to the entire powerset of the Languages set

   (b) Favorite is a single non-atomic element of the powerset of the Languages set. Not the entire powerset of Languages.

4. No, because the element of the Language set {Lua, Groovy, C} is not a set of sets and therefore cannot be a member of the powerset of the Language set. Having the set {Lua,Groovy,C} in the power set of Languages would imply that the atomic variable Lua, Groovy and C are contained within the Languages set, which is not the case for Lua.

5. Yes, because {{Lua,Groovy,C}} is an element of the powerset of Languages where the superset only contains the element {Lua,Groovy, C}, and not the individual atomic variables.

6. (a) The variable can be either atomic or non-atomic, depending on what element of

Languages it is assigned to. This is because Languages holds both atomic and non-atomic variables.

   (b) Non-Atomic. This variable represents a set of elements

7. Library: PLanguages

8. No, because the set containing the empty set (‘{{}}’) is not an element of the powerset but the empty set (‘{}’) would be.

9. Refer to problem4-9.lsp in /code

Problem 5: Ordered structures

1.

   Enqueue(Q, T):

   cons(T , Σ1)

   Dequeue(Q):

   while Σ1 is not empty:

   cons(head(Σ1) , Σ2)

   Σ1’ = tail(Σ1)

   If Σ2 is empty:

   return null

   Else:

   Σ1’ = Σ2

   Σ2 = cons(nil , ⟨⟩)

   Return head(Σ1)

2. Refer to file queue-adt.lsp in /code

Problem 6: Binary relations, functions and orderings

Part 1

1. Given binary relation R : "is of type" in the domain of types in the Java API, we find that:

   1 - R is reflexive, $\forall a \in A : aRa$, meaning for any type a, a is of type a. This is true because, in the Java API, any type is considered to be a type of itself;

   2 - R is anti-symmetric, $\forall a, b \in A : (aRb \land bRa) \rightarrow a = b$, meaning for any type a and b, if a is of type b and b is of type, then a = b. In the Java API, for an element to be of a specific type, it must be an element of that type, or an element of a subtype of that type. This means that for two types to be types of each other, they must be the same type.

   3 - R is transitive, $\forall a, b, c \in A : (aRb \land bRc) \rightarrow aRc$, meaning for any type a, b and c, if a is of type b, and b is of type c, then a is of type c. As stated previously, for an element to be of a specific type, it must be an element of that type, or an element of a subtype of that type. So, if a is of type b, and b is of type c, we can come to the conclusion that a is of type c.

   Therefore, we find that R is a partial order.

2. Given the set of vertices V1 and edges E, we can prove that (V1, R) is a poset:

   1 - From the given edges, there are explicit examples of the reflexive property. However, in the Java API context, reflexivity is implied, as any type is considered to be a type of itself.

   2 - In the given edges, there are no examples of two types relating to each other, and not being the same type, therefore, the conditions for anti-symmetry are satisfied.

   3 - Given the context of the Java API, (V1,R) can be considered transitive because of inheritance. For example, in the given edges, we find that NavigableMap is of type SortedMap, and that SortedMap is of type Map, therefore, considering the context, NavigableMap is also of type Map.

3.

Part 2

1. 1 - The binary relation is subset of is reflexive, $\forall$a $\in$ A : aRa, since every set is a subset of itself.

   2 - It is antisymmetric, $\forall$a, b $\in$ A : (aRb $\wedge$ bRa) $\rightarrow$ a = b, since any two sets that are subsets of each other must be the same set.

   3 - It is also transitive, $\forall$a, b, c $\in$ A : (aRb $\wedge$ bRc) $\rightarrow$ aRc, because if a set S1 is a subset of a set S2, and S2 is a subset that a set S3, then S1 is a subset of S3. For example, S1 = $\{1, 2\}$, S2 = $\{1, 2, 3\}$ and S3 = $\{1, 2, 3, 4\}$, S1 is a subset of S2, and S2 is a subset of S3, and S1 is also a subset of S3.

2. P(V2) = $\{\{\}, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}\{a, b, c\}\}$

   1 - The power set of V2 is indeed reflexive since all sets within P(V2) are subsets of themselves.

   2 - It is anti-symmetric since there are no 2 distinct elements that are subsets of each other, or (a, b) $\in$ A $\wedge$ a $\neq$ b $\rightarrow$ (b, a) $\notin$ R.

   3 - It is transitive since $\forall$a, b, c $\in$ A : (aRb $\wedge$ bRc) $\rightarrow$ aRc. This can be seen in P(V2) from $\{a\}$, $\{a, b\}$ and $\{a, b, c\}$, where $\{a\}$ is a subset of both $\{a, b\}$ and $\{a, b, c\}$, and $\{a, b\}$ is a subset of $\{a, b, c\}$.

Part 3

1. - Map is indeed a function, since every element in the domain maps to at most 1 element in the co-domain.

2. - It is a partial function, since not every element in the domain participated in the mapping. map : V1 $\not\rightarrow$ P(V2)

3. - It is not injective since some elements in the co-domain are mapped to twice, namely $\{a, b, c\}$. The definition for an injective function is $\forall$x, y $\in$ V1 : x $\neq$ y $\rightarrow$ f (x) $\neq$ f

(y), which is not followed in this case.

4. - It is not surjective since not every element in the co-domain is mapped to by at least one element in the domain. The definition for a surjective function is $\forall$z $\exists$ P(V2), $\exists$x $\in$ V1 : f (x) = z, which is not the case.

5. - Since the function is neither injective nor surjective, it is not bijective.

6. - The function is order-preserving since all predecessor relationships between elements in the domain are preserved by their images in the codomain. In other words x $\prec$ y in V1 implies f (x) $\prec$ f (y) in P(V2). In this case, we have NavigableMap $\rightarrow$ c and TreeMap $\rightarrow$ $\{\}$, where TreeMap $\prec$ NavigableMap in V1 implies $\prec$ c in P(V2).

7. - The function is not order-preserving since not all predecessor relationships between elements in the codomain are reflected by their pre-images in the domain. In other words, we say a function is order reflecting if f (x) $\prec$ f (y) in P(V2) implies x $\prec$ y in V1. In this case, we have $\{c\} \prec \{a, b, c\}$, which is not reflected by the images, meaning we don't have NavigableMap $\prec$ AbstractMap.

8. - The function is not order-embedding since it is not both order-preserving and order-reflecting.

9. - The function is not isomorphic since it is neither order-embedding nor surjective.

Problem 7: Construction techniques

1.
```
compress (T) is
 comp_list = ⟨⟩
 for all i in T
  if(head(comp_list) != i)
   consR(com_list, i)
 return comp_list
```

2.
```
compress (T) is
 if(T = ⟨⟩) then
  return ⟨⟩
 if(head(T) = head(tail(T))) then
  return compress(tail(T))
 else
  return cons(head(T), compress(tail(T)))
```

3.
```
compress(⟨a, a, b, b, c, a⟩) = compress(⟨a,b,b,c,a⟩)
       = cons(a, compress(⟨b,b,c,a⟩))
       = cons(a, compress(⟨b,c,a⟩))
       = cons(a, cons(b, compress(⟨c,a⟩)))
       = cons(a, cons(b, cons(c, compress(⟨a⟩))))
       = cons(a, cons(b, cons(c, cons(a, compress(⟨⟩)))))
       = cons(a, cons(b, cons(c, cons(a, ⟨⟩))))
       = cons(a, cons(b, cons(c, ⟨a⟩)))
       = cons(a, cons(b, ⟨c ,a⟩))
       = cons(a, ⟨b ,c ,a⟩)
       = ⟨a, b ,c ,a⟩
```

4. Refer to file compress.lisp in /code