Homework: Optional Assignment 3

Due-date: Apr 30 at 11:59pm
Submit online on Canvas

Homework must be individual's original work. Collaborations and of any form with any students or other faculty members are not allowed. If you have any questions and/or concerns, post them on Piazza and/or ask 342 instructor or TAs.

This is an optional assignment. If you do this assignment, the points you get for this assignment will be considered as extra-credits, and will add at most 8 *points to your overall grade.*

Learning Outcomes

- Knowledge and application of Functional Programming
- Ability to understand graphical modeling
- Ability to design software following requirement specifications (preference logic)

Questions

One of the central challenges in solving many optimization problems is to appropriately manage choices (selecting movies/shows on Netflix, buying products at the supermarket or at Amazon, getting a flagship phone or a phone-plan, deploying security measures to counter system vulnerabilities, etc.) The choices are described in terms of properties of attributes and we, often, have preferences over these properties.

In this problem, we will assume that our choices are described using just three attributes. Each attribute can take up values from its domain and the user has preferences between these values. Preferences between values of attribute i is described by the relation $\succ_i$ ($x \succ_i x'$: for the values of the attribute i, x is preferred to x'). The relation is transitive and asymmetric. Consider two choices c and c' such that c is described by the values of three attributes: x (value of attribute 1), y (value of attribute 2) and z (value of attribute 3), and c' is similarly described by x', y' and z'. The choice c is preferred to choice c' (denoted by $c \succ c'$) if and only if one of the following conditions hold

1. there exists an attribute such that the corresponding valuation for c is preferred to that for c', and all other attribute-values for c and c' are identical. We call this a ***flip***; OR

2. there exists a sequence of choices $c_0, c_1, c_2, \ldots, c_n$ such that $\forall i \in [0, n-1] : c_i \succ c_{i+1}$ and $c_0 = c$, $c_n = c'$. We call this a flipping sequence.

Given such a preference relation $\succ$ among the **possible choices**, our goal is to order the **available choices** in a specific way—we will refer to this as *weak order*. Let $A = \{a_1, \ldots, a_n\}$ be a set of n available choices, then a weak order of these choices is presented by a sequence of subsets of A:

$$L_1, L_2, \ldots, L_k$$

such that

1. $L_1 \cup L_2 \cup \ldots \cup L_k = A$
2. $\forall i, j \in [1, k] : (i \neq j) \Rightarrow (L_i \cap L_j = \emptyset)$
3. $\forall i \in [1, k] : \forall x, y \in L_i : x \not\succ y \ \wedge \ y \not\succ x$
4. $\forall i \in [1, k-1] : \forall y \in L_{i+1} \exists x \in L_i : x \succ y$

In other words, there are no available choices preferred to any available choices in L_1. Intuitively, the available choices in L_1 can be viewed as the set of most preferred available choices. If for some reason (beyond the ones expressible by preferences, e.g., some choice becomes unavailable at the time of selection), none of the L_1 choices are viable selections, then the user can review the choices in L_2 (intuitively, L_2 choices are next best available choices after choices in L_1). Such a strategy of review-reject/select in a level-by-level basis proceeds until the user identifies an available choice to select (or gives up and refines his/her preferences).

Example Scenario. As an example, we will consider a scenario for car-buying. Any car (a choice) is described by three attributes: color, engine-type and maker. The attributes take values from respective domain: color can be blue, black, red, white; engine-type can be electric, hybrid, four-cylinder and six-cylinder; maker can be tesla, bmw, skoda, alfa. Therefore, there are 4^3 *possible choices*. As an user, you can assert preferences between the values of the attributes. For instance, you have asserted

1. For color attribute:
   (a) black is preferred to red, blue and white
   (b) blue is preferred to red
   (c) white is preferred to blue and red
2. For enginetype
   (a) electric is preferred to four- and six-cylinder
   (b) hybrid is preferred to four- and six-cylinder
   (c) four-cylinder is preferred to six-cylinder
3. For maker
   (a) tesla is preferred to skoda, alfa and bmw
   (b) bmw is preferred to skoda and alfa
   (c) alfa is preferred to skoda

Consider that you have 8 available choices:
(red electric tesla)
(black hybrid, bmw)
(blue electric bmw)

(red hybrid bmw)
(red four alfa)
(blue electric tesla)
(black four alfa)
(black electric skoda)

Here is an example that presents $\succ$-relation between two available choices.

(black hybrid bmw) $\succ$ (red four alfa) because
(black hybrid bmw) $\succ$ (red hybrid bmw) — black $\succ_1 red$
(red hybrid bmw) $\succ$ (red four bmw) — hybrid $\succ_2$ four
(red four bmw) $\succ$ (red four alfa) — bmw $\succ_3$ alfa

On the other hand, (black electric skoda) is $\not\succ$ (blue electric bmw) because (black electric skoda) $\succ$ (blue electric skoda) but (blue electric skoda) $\not\succ$ (blue electric bmw). On similar note, (blue electric bmw) is $\not\succ$ (black electric skoda).

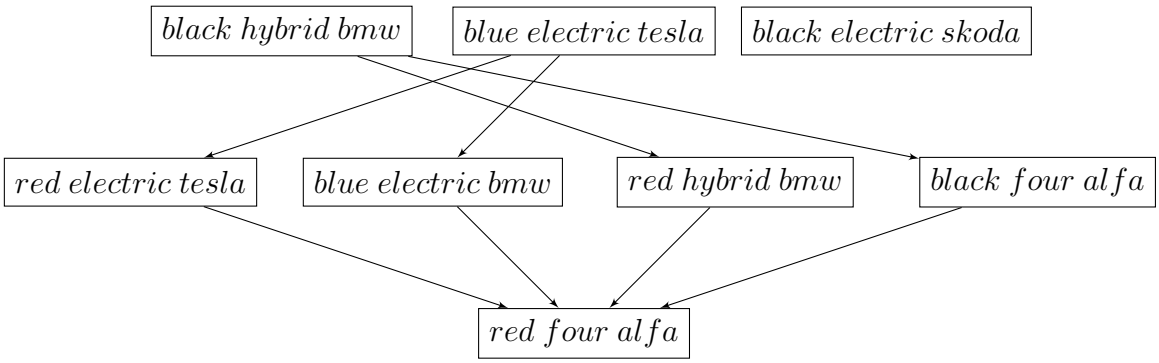
Your objective is to obtain a weak order of these available choices. The weak order of the available choices is

L1: (black hybrid bmw) (blue electric tesla) (black electric skoda)

L2: (red electric tesla) (blue electric bmw) (red hybrid bmw) (black four alfa)

L3: (red four alfa)

In the following figure, we have graphically represented the $\succ$ relation (direction of arrow indicates $\succ$) over the available choices between successive levels.



Racket Encoding for your assignment. For each attribute, the preferences between the values will be represented as a list. Each element in the list is also list of the form `(attribute-value list-of-attribute-values)`. For instance, the preferences over the attribute 1 (color) in our example will be defined as follows:

```
(define attr1
   '(
      (black (red blue white)) ;; black is preferred to red, blue and white
      (blue  (red))            ;; blue is preferred to red
      (white (blue red))       ;; white is preferred to blue and red
    ))
```

Similarly,

```
(define attr2
  '(
    (electric (four six))
    (hybrid (four six))
    (four (six))))

(define attr3
  '(
    (tesla (skoda alfa bmw))
    (bmw (skoda alfa))
    (alfa (skoda))))
```

Each available choice will be defined as a list of three elements, where the first element is a value for attribute 1, the second element is a value for attribute 2 and the third element is a value for attribute 3. In our example, the available choices is defined as follows:

```
(define carlist
  '( (red electric tesla)
     (black hybrid bmw)
     (blue electric bmw)
     (red hybrid bmw)
     (red four alfa)
     (blue electric tesla)
     (black four alfa)
     (black electric skoda)))
```

You are required to write a function `weakorder` which with four formal parameters: list of available choices and preferences with respect to three attributes, and outputs a list. Each element in the output list is a list itself (corresponding to a level). The first element in the output list is the list of available choices at level 1, the second element in the output list is the list of available choices at level 2, and so on. For our current example, the call-pattern and the output will be

```
> (weakorder carlist attr1 attr2 attr3)
'( ( (black hybrid bmw)
     (blue electric tesla)
     (black electric skoda)
   )
   ( (red electric tesla)
```

```
    (blue electric bmw)
    (red hybrid bmw)
    (black four alfa)
  )
  ( (red four alfa)
  )
)
```

The ordering of available choices at each level does not matter.

Programming Rules

- You are required to submit one file hwopt3-⟨net-id⟩.rkt[1]. The file must start with the following.

  ```
  #lang racket
  (provide (all-defined-out))
  ```

  In the above, the tests.rkt will be used as an input file for our test cases.

- You are **only allowed** to use functions, if-then-else, cond, basic list operations (**you can use the built-in append, sort function**), operations on numbers. No imperative-style constructs, such as begin-end or explicitly variable assignments, such as get/set are allowed. If you do not follow the guidelines, your submission will not be graded. If you are in doubt that you are using some construct that may violate rules, please contact instructor/TA (post on Piazza).

- You are expected to test your code extensively. If your implementation fails any assessment test, then points will be deducted. Almost correct is equivalent to incorrect solution for which partial credits, if any, will depend only on the number of assessment tests it successfully passes.

- The focus is on correct implementation. In this assignment, we will not assess the efficiency; however, avoid re-computations and use tail-recursion, if possible.

Guidelines

1. **Please make sure your submission does not have syntax errors.**
2. **Please remove the trace-directives.**
3. **Please remove any test-code.**
4. **Please include `(provide (all-defined-out))`.**
5. Learn how to use Racket. If you have not written some definitions in Racket and have not reviewed the solutions to any/some exercises, then it is likely to be difficult to complete this assignment.

[1] Your netid is your email-id and please remove the angle brackets, they are there to delimit the variable net-id.

6. Starting two days before the deadline to do the above is likely to make it impossible for you to complete this assignment.

7. As always, map out the functions you need to write, write comments that include the specification of the functions you are writing, test each function extensively before using it in another function.

Postscript. It is not necessary to read this for successfully completing the assignment. This section is intended to provide with some additional context/knowledge. So far you have worked on functionally solving problems that relate to language semantics, job scheduling, simple classification and model checking.

In this optional assignment, you are computing the semantics of a special type of logic: preference logic, in particular, a simplified version of logic whose semantics is captured by a preference network. Such logic is specifically used to order choices in scenarios where the preferences are not expressible quantitatively. For instance, it may not be possible to describe quantitatively by how much I prefer the color black to red. Furthermore, in some cases, it not possible to present a specific ordering: I may not know whether I prefer blue to white or white to blue, therefore, (total) ordering them is impossible. Such scenarios are typical in almost all decision theoretic problems (project management/development, security analysis, public policy such health-care, city-planning, etc.), which makes qualitative preference reasoning an important sub-field of AI.