Problem Set 10

OUT: Monday 3/23/2015, 9pm EST
DUE: Monday 3/30/2015, 9pm EST

Last updated: Mon, 16 Mar 2015 16:14:24 -0400

Preliminaries

- You must use the `Intermediate Student with lambda` language to complete this assignment. Select it via the `Choose Language...` menu located at the bottom-left of the DrRacket window.
- Put all your solution files in a directory named `set10` in your repository.
- Download extras.rkt to this directory (right-click and choose "Save As"; don't copy and paste) and commit it as well.
- Use `begin-for-test` and `rackunit` to define your examples and tests.
- Don't forget to tell us how many hours you spent working on the assignment. This should be a global variable called `TIME-ON-TASK` in each file. For example:

```
(define TIME-ON-TASK 10.5) ; hours
```

- So each solution file must have at least the following at the top:

```
(require "extras.rkt")
(require rackunit)
(define TIME-ON-TASK <number-of-hours-you-spent>)
```

- After you've submitted your solution, use a web browser to go to https://github.ccs.neu.edu/ and check that your repository contains the following files:
  - `set10/sched-with-unavail.rkt`
  - `set10/sched-general.rkt`
  - `set10/sched-with-avail.rkt`
  - `set10/extras.rkt`
- **Git Commit Requirement**: For this assignment, you must have at least three well-labeled git commits (including the final commit). A well-labeled git commit accurately and succinctly describes the changes since the previous commit. Something like `"commit2"`, or `"home work 3"` is not an acceptable git commit label. Failure to meet this requirement will result in loss of points.
- Be sure to state a proper termination argument for any functions that utilize generative recursion.

1 Code Walk Scheduler, Part 1

1.1 Additional Preliminaries

Save your solutions for this problem to a file named `sched-with-unavail.rkt`.

Run the following expression (you must have required extras.rkt) to check that your file is properly named and is in the proper directory:

```
(check-location "10" "sched-with-unavail.rkt")
```

Add these additional provides at the top of your file (below the requires), so that we can test your solution:

```
(provide schedule/unavail-ok?)

(provide schedule/unavail)
```

1.2 Problem Description

The PDP staff needs your help scheduling codewalks. For this assignment, you will implement a codewalk scheduling algorithm.

A codewalk schedule is an assignment of students to codewalk times, as described in the data definitions below. Complete the rest of the Data Design step for these data definitions. Add any additional data definitions as you see fit.

```
; A CodeWalks is a ListOf<CodeWalk>
; Represents a codewalk schedule.

; A CodeWalk is a (make-codewalk Time ListOf<StudentID> PosInt)
; Represents a codewalk with time, assigned students, and a max capacity.
(define-struct codewalk (time students max))
(define CW-TUE135 (make-codewalk TUE135 empty 1))
(define CW-TUE325 (make-codewalk TUE325 empty 1))

; A Time is a Symbol
; Represents a day of week and time.
(define TUE135 '1:35pmTues)
(define TUE325 '3:25pmTues)

; A StudentID is a Symbol
; Represents a student (or pair) via their ccs ID(s).

; A StudentUnavail is a (make-student StudentID Preferences)
(define-struct student (id prefs))
; Represents a student and their unavailable times.

; A StudentUnavails is a ListOf<StudentUnavail>

; A Preferences is a ListOf<Time>
; Represents a list of code walk times.
```

Tasks:

1. Implement `schedule/unavail-ok?`, a predicate that checks whether a schedule satisfies a list of student preferences. A schedule is valid according to some preferences if:

   a. each code walk is not over capacity,

   b. each student is scheduled exactly once,

   c. each student is scheduled in an acceptable time slot according to their preferences.

```
; schedule/unavail-ok? : StudentUnavails CodeWalks -> Boolean
; Returns true if cws is a valid schedule according to the given
; student preferences.
(define (schedule/unavail-ok? students cws) ...)
```

2. Implement the `schedule/unavail` function.

```
; schedule/unavail : StudentUnavails CodeWalks -> Maybe<CodeWalks>
; Creates a codewalk schedule by assigning students to cws,
; while satisfying students' constraints.
; Returns #f if no schedule is possible.
; Strategy: ???
(define (schedule/unavail students cws) ...)
```

2 Code Walk Scheduler, Part 2

2.1 Additional Preliminaries

Save your solutions for this problem to a file named `sched-general.rkt`.

Run the following expression (you must have required extras.rkt) to check that your file is properly named and is in the proper directory:

```
(check-location "10" "sched-general.rkt")
```

Add these additional provides at the top of your file (below the requires), so that we can test your solution:

2.2 Problem Description

Tasks:

- Add the following data definitions:

```
; A StudentAvail is a (make-student StudentID Preferences)
; Represents a student and their available times, most-preferred first.
; An unlisted time means the student is unavailable.

; A Student is one of:
; - StudentUnavail
; - StudentAvail
```

```
; A StudentAvails is a ListOf<StudentAvail>
; A Students is a ListOf<Student>
```

Don't worry about templates for these data definitions, since none of your functions should have to decompose the `Student` itemization.

- Abstract your program such that it's easy to implement versions of the scheduling predicate and algorithm from part 1 for either version of student. You will almost certainly need to user higher-order functions, à la `map`, `filter`, and friends.

  As one possible example, you might define a function:

  ```
  ; schedule/general : Students CodeWalks ... -> Maybe<CodeWalks>
  (define (schedule/general students cws ...) ...)
  ```

  Then `schedule/unavail` would simply be a call to `schedule/general`. However, you may also choose to abstract in a different manner.

  Either way, if you've been following the concepts in the course, and have organized your code in a clean manner, this is an extremely easy refactoring of no more than a few lines of code and should take no more than a few minutes.

  NOTE: Your abstractions should not come at the expense of code readability. All your signatures and purpose statements must remain valid and clear. You will fail this assignment if this is not the case.

  (If you are struggling with this part of the assignment, you may try to first complete parts 1 and 3, before coming back to part 2. If you do this, you should observe a large amount of code duplication between parts 1 and 3, and the possible abstractions may be more evident.)

- Split your code into two files:

  - `sched-with-unavail.rkt`: All code in this file only handles the first kind of student, `StudentUnavail`. This file still implements the `schedule/unavail` `schedule/unavail-ok?` functions from part 1.

  - `sched-general.rkt`: All code in this file handles the general `Student` data definition. For example, you might put the `schedule/general` function mentioned above in this file. This file should be `require`d by `sched-with-unavail.rkt`. You may find it helpful to add `(provide (all-defined-out))` at the top of this file (make sure you have the most recent "extras.rkt" file). However, no functions from this file will be accessed by our test suite.

  If you have followed the concepts from this course, `sched-general.rkt` file should contain the bulk of the code (not counting tests).

  Testing Note: If all the code was properly tested before splitting the files, it's acceptable if a few of the generalized functions in `sched-general.rkt` do not have full test coverage, since these functions are expected to be instantiated and tested in `sched-with-unavail.rkt`.

3 Code Walk Scheduler, Part 3

3.1 Additional Preliminaries

Save your solutions for this problem to a file named `sched-with-avail.rkt`.

Run the following expression (you must have required extras.rkt) to check that your file is properly named and is in the proper directory:

```
(check-location "10" "sched-with-avail.rkt")
```

Add these additional provides at the top of your file (below the requires), so that we can test your solution:

```
(provide schedule/avail-ok?)

(provide schedule/avail)

(provide avg-choice)
```

3.2 Problem Description

Implement a predicate and scheduling algorithm for the second kind of student, `StudentAvail`. `sched-with-avail.rkt` should `require` and re-use functions from `sched-general.rkt`. If you have properly organized your code, implementing these functions should take no more than a few lines of code.

Tasks:

1. Implement `schedule/avail-ok?`, a predicate that checks whether a schedule satisfies a list of student preferences. A schedule is valid according to some preferences if:
   a. each code walk is not over capacity,
   b. each student is scheduled exactly once,
   c. each student is scheduled in an acceptable time slot according to their preferences.

```
; schedule/avail-ok? : StudentAvails CodeWalks -> Boolean
; Returns true if cws is a valid schedule according to the given
; student preferences.
(define (schedule/avail-ok? students cws) ...)
```

2. Implement the `schedule/avail` function.

```
; schedule/avail : StudentAvails CodeWalks -> Maybe<CodeWalks>
; Creates a codewalk schedule by assigning students to cws,
; while satisfying students' constraints.
; Returns #f if no schedule is possible.

; Strategy: ???
(define (schedule/avail students cws) ...)
```

3. Implement `avg-choice`, which computes the average of the rank of the codewalks assigned to each student. For example, if each student is assigned their first choice,

then `avg-choice` returns 1. This roughly indicates how well the scheduling algorithm performs.

```
; avg-choice : StudentAvails CodeWalks -> PosReal
; WHERE: (schedule/avail-ok? students cws) = #t
(define (avg-choice students cws) ...)
```

4. Tinker with your implementation of `schedule/avail`, with the goal of producing a schedule that maximizes `avg-choice`. We will apply your scheduler to a dataset of (Boston) student-submitted preferences. Any submission that beats our reference implementation will be rewarded some (to-be-determined) number of bonus points.