Учебное пособие по выполнению задания «Доктор»

Набор упражнений в рамках общего задания «Доктор» первоначально был составлен канд. физ.-мат. наук Черновым А. В. более десяти лет тому назад. Предложенный им текст доступен онлайн [pdf]. Затем в течение ряда лет задание обновлялось канд. физ.-мат. наук Малышко В. В.. Результаты этих обновлений перед Вами.

Оглавление

Введение

«Доктор» (или ELIZA) -- это название программы, созданной Джозефом Вейценбаумом. Эта программа имитирует (или пародирует) психоаналитика, ведущего диалог с пациентом. Программа принимает реплики пациента (в виде списков символов) и генерирует ответные реплики (также в виде списков). Для простоты сокращена пунктуация в записи реплик. Исходный код ELIZA на языках LISP-семейства можно найти в Сети: [html], [html].

В рамках выполнения практического задания Вы создадите собственную версию «Доктора», которая основана на некоторых (но не всех!) идеях, лежащих в основе исходной программы.

Упражнения по «Доктору» делятся на 4 блока:
1-й блок -- упражнения с 1 по 4. Максимальный балл -- 5.
2-й блок -- упражнения 5 и 6. Максимальный балл -- 7.
3-й блок -- упражнение 7. Максимальный балл -- 8.

Рекомендуется сдавать упражнения блоками. Датой сдачи блока считается дата последнего сданного упражнения из блока, при условии что код прислан вовремя. Во время сдачи программы будьте готовы ответить на вопросы по коду, а также по материалам лекций. Например, какой процесс порождает та или иная функция: итеративный или рекурсивный, как работает та или иная спецформа или функция (в том числе, функция, написанная Вами). Может быть предложено оценить эффективность фрагмента Вашего кода и улучшить её, переписав фрагмент.

После сдачи любого из блоков «Доктора» следует прислать код лектору. Если код не прислан в течение 7 дней после сдачи, по заданию аннулируется отметка о сдаче.

Версии программы 1-3 блоков следует составлять на подмножестве языка scheme/base (начинайте свой код с директивы #lang scheme/base). В них использование мутаторов (присваиваний и т. п.), мутируемых структур данных Racket, запрещено. Составленная в ходе выполнения упражнений, программа должна быть показана лектору в ходе офлайновой сдачи. При переходе от начальных упражнений к последующим код следует дописывать так, чтобы функциональность программы расширялась (то, что было раньше, не портить). Не следует сдавать несколько разных версий программы, каждая из которых решает только одно из упражнений.

Предполагается, что прошлые годы учёбы привили Вам навык оформления кода. Сдаваемый код должен быть оформлен должным образом: быть читаемым, сопровождаться содержательными и актуальными комментариями на русском языке.

Блок 1. Упражнения с 1 по 4

Упражнение 1. Заготовка doctor.rkt

В документах ВК-группы [html] найдите заготовку кода «Доктора» и скачайте её. На сайте [html] найдите дистрибутив среды Racket и установите её себе на компьютер. Запустите IDE Racket и откройте заготовку кода «Доктора». Запустите код в интерпретаторе (Ctrl+R или кнопкой Run). Проверьте, как он работает. Например, вводите реплики пациента из листинга (реплики отмечены **, но ** не являются частью реплик):

```
> (visit-doctor 'ivan)
Hello, ivan!
(what seems to be the trouble?)
**(everyone hates me)
(you feel that everyone hates you)
**(whenever i enter a room no one will look me in the eye)
(please go on)
**(they laugh at me behind my back)
(you seem to think they laugh at you behind your back)
**(they get together and talk about how they hate me)
(please continue)
**(even my mother hates me)
( that even your mother hates you)
**(even when i was a baby she said i was ugly and stupid)
(many people have the same sorts of feelings)
```

```
**(she hit me all the time)
(please continue)
...
```

«Доктор» отвечает на реплики пациента двумя способами, реализованными в нём. Первый способ заключается в замене в реплике пользователя всех местоимений первого лица (i, me, my, ...) на соответствующие местоимения второго лица (you, you, are, ...). Местоимения второго лица в реплике пользователя заменяются на местоимения первого. К полученной таким образом фразе спереди дописывается некоторое общее замечание, вроде `why do you say that` или `you seem to think that`.

Второй метод проще первого. Реплика пациента при построении ответа не используется. В качестве ответа «Доктор» выдаёт одну из заготовленных общих фраз, выбранную случайно: `please continue` или `many people have the same sorts of feelings`.

Какой из двух методов будет использован, решает случай. С равной вероятностью выбирается один из двух.

«Доктор» запускается вызовом его основной функции -- `visit-doctor` с указанием имени пациента в качестве значения параметра. В этой функции сначала выводится приветствие пациента, задаётся вопрос, приглашающий к обмену репликами, а затем вызывается функция `doctor-driver-loop`, реализующая цикл приёма реплик пациента и генерации ответов «Доктора».

```
(define (visit-doctor name)
  (printf "Hello, ~a!\n" name)
  (print '(what seems to be the trouble?))
  (doctor-driver-loop name)
)
```

В цикле происходит печать приглашения -- ** -- и чтение реплик пациента, каждая из которых является списком символов (пользователю следует вводить их в скобках). Если пациент говорит `(goodbye)`, происходит выход из цикла и завершение работы «Доктора». В противном случае, порождается ответная реплика в соответствии с одной из двух стратегий, описанных выше.

```
(define (doctor-driver-loop name)
    (newline)
    (print '**) ; доктор ждёт ввода реплики пациента, приглашением к которому является **
    (let ((user-response (read)))
      (cond
            ((equal? user-response '(goodbye)) ; реплика '(goodbye) служит для выхода из цикла
             (printf "Goodbye, ~a!\n" name)
             (print '(see you next week)))
            (else (print (reply user-response)) ; иначе Доктор генерирует ответ, печатает его и продолжает цикл
                  (doctor-driver-loop name)
             )
       )
     )
)
```

Генерацию ответной реплики осуществляет функция `reply`, принимающая реплику пациента в качестве значения параметра.

```
(define (reply user-response)
      (case (random 2)  ; с равной вероятностью выбирается один из двух способов построения ответа
          ((0) (qualifier-answer user-response)) ; 1й способ
          ((1) (hedge))  ; 2й способ
      )
)
```

Вторая стратегия построения ответов реализована функцией `hedge`. Она случайно выбирает из списка заготовленных реплик один элемент.

```
(define (hedge)
       (pick-random '((please go on)
                      (many people have the same sorts of feelings)
                      (many of my patients have told me the same thing)
                      (please continue)
                      )
        )
)
```

Случайный выбор одного из элементов списка реализует функция `pick-random`.

```
(define (pick-random lst)
  (list-ref lst (random (length lst)))
)
```

Первая стратегия построения ответов реализована функцией `qualifier-answer`, получающей реплику пациента в качестве значения параметра. В ней случайно выбирается одно из заготовленных начал ответа. Окончанием ответа является реплика пользователя, в которой заменены лица.

```
define (qualifier-answer user-response)
       (append (pick-random '((you seem to think that)
                              (you feel that)
                              (why do you believe that)
                              (why do you say that))
               )
               (change-person user-response)
       )
)
```

Замену лиц осуществляет функция `change-person`. Она вызывает более универсальную функцию `many-replace`, получающую список замен и список, в котором следует произвести замены. Она проверяет каждый элемент изменяемого списка, на возможность его замены. Если замена возможна (элемент найден в списке замен, то она производится, а окончание ответа находится при помощи рекурсивного вызова. Список замен является ассоциативным списком. Он составлен из подсписков, каждый из которых начинается с элемента-ключа и продолжается элементом-значением. Поиск подсписка по ключу выполняет стандартная функция `assoc`. Обратите внимание, что в `if` используется результат вызова `assoc`, который не всегда будет логическим значением. Убедитесь, что Вы верно понимаете как работают `if` и `assoc`.

```
(define (many-replace replacement-pairs lst)
        (cond ((null? lst) lst)
              (else (let ((pat-rep (assoc (car lst) replacement-pairs))) ; Доктор ищет первый элемент списка в ассоциативном списке замен
                      (cons (if pat-rep (cadr pat-rep) ; если поиск был удачен, то в начало ответа Доктор пишет замену
                                (car lst) ; иначе в начале ответа помещается начало списка без изменений
                            )
                            (many-replace replacement-pairs (cdr lst)) ; рекурсивно производятся замены в хвосте списка
                      )
                    )
              )
        )
)
```

Задание упражнения 1. Измените функции `qualifier-answer` и `hedge`, добавив в каждую не менее трёх новых заготовленных фраз-реплик и/или фраз, с которых начинается ответ с заменой лица.

Упражнение 2. Рекурсивные и итеративные процессы

Рассмотрим код функции `many-replace`. В ней с рекурсивным вызовом связаны остаточные вычисления. Следовательно, процесс, порождаемый при вычислении вызова этой функции, -- рекурсивный. Можно переписать `many-replace` так, чтобы процесс стал итеративным и вычисления занимали меньше памяти за счёт хвостовой рекурсии.

Задание упражнения 2. Напишите новую версию функции `many-replace` с хвостовой рекурсией и вызовите её в теле `change-person`. Составьте код нового `many-replace` без определения локальной вспомогательной функции, а с использованием "именованного" `let`. Далее всякий раз, когда для реализации итеративного процесса понадобится локальная вспомогательная функция, используйте вместо неё "именованный" `let`. Новая версия функции должна быть эффективной. Если Вы планируете использовать `append` для "сборки" результата, то оцените сложность решения. Как получить ответ без `append`?

Упражнение 3. Функции высших порядков и списки

Рассмотрим ещё раз реализацию `many-replace`. Она осуществляет отображение списка `lst` в список-результат. Общую схему подобной обработки списка реализует функция `map`.

Задание упражнения 3. Напишите ещё одну версию функции `many-replace` и замените вызов в теле `change-person`. Сделайте так, чтобы тело новой версии состояло только из вызова `map`. Дополнительную функцию не определяйте отдельно, а заведите как анонимную -- результат вычисления спецформы `lambda`. Переписанная функция должна быть столь же эффективной, как результат выполнения упражнения 2. В ходе дальнейшей работы над «Доктором» всякий раз, когда описываете обработку списка, сделайте это с помощью подходящей функции высшего порядка (`map`, `foldl`, `foldr`, `filter`, `andmap`, `ormap` ...). В модуле, подключаемом директивой `(require racket/list)` есть дополнительные функции высшего порядка для работы со списками (`filter-map`, `count`, `append-map`, `filter-not`, `argmin`, `argmax`, `remf`, `remf*`, ...), имейте это в виду.

Упражнение 4. 3-я стратегия генерации ответов

Двух способов генерации ответов мало. Добавим третий. «Доктор» может запоминать все реплики пациента и в ходе беседы возвращаться к сказанному пациентом ранее. В этом случае реплика «Доктора» будет начинаться со слов `earlier you said that`, а затем будет следовать одна из предыдущих реплик пациента, в которой выполнена замена лица. Например, если из предыдущих реплик пациента выбрана реплика (`you are not being very helpful to me`), то по ней будет построен ответ (`earlier you said that i am not being very helpful to you`).

Задание упражнения 4. Измените программу таким образом, чтобы `doctor-driver-loop` сохранял список всех реплик пользователя. Замечание: не используйте присваивания (мутаторы вроде `set!`). В этом нет необходимости. Новая версия функции `reply` будет выбирать одну стратегию из трёх. Новую стратегию реализуйте как отдельную функцию `history-answer`. Обратите внимание, что `hedge` и `qualifier-answer` можно применять всегда, а `history-answer` -- только при наличии предыдущих реплик. Когда пациент вводит первую реплику, эта стратегия не применима. Учтите это в реализации. Для случайного выбора из трёх альтернатив следует взять случайное число от 0 до 2: (`random 3`), и использовать его как номер выбранной стратегии, считая, что они нумеруются с нуля. Для вызова стратегии по выбранному номеру подходит спецформа `case`.

Блок 2. Упражнения 5 и 6

Упражнение 5. Многопользовательский «Доктор»

Текущая версия программы умеет работать только с одним пациентом, имя которого задаётся в вызове функции `visit-doctor`. Когда пациент говорит `goodbye`, `visit-doctor` возвращает управление интерпретатору. Измените программу таким образом, чтобы «Доктор» автоматически переходил к приёму следующего пациента после прощания с предыдущим. Предусмотрите способ завершения работы многопользовательского «Доктора» или после использования стоп-слова в качестве имени очередного пациента, или при исчерпании количества принимаемых пациентов. Стоп-слово и максимальное количество пациентов передаются в обновлённый `visit-doctor` как значения его параметров. Например, `(visit-doctor 'suppertime 3)` может завершиться либо если трое пациентов обслужено, либо если введено имя пациента `suppertime`. Для ввода имени очередного пациента можете воспользоваться функцией `ask-patient-name`

(заметьте, что именем пациента считается первый элемент списка, введённого пользователем):

```
(define (ask-patient-name)
 (begin
  (println '(next!))
  (println '(who are you?))
  (print '**)
  (car (read))
 )
)
```

Теперь сессия работы «Доктора» может выглядеть следующим образом:

```
> (visit-doctor 'suppertime 3)
(next!)
(who are you?)
**(hal abelson)
(hello, hal)
(what seems to be the trouble?)
**(everyone taking 6.001 hates me)
(why do you say everyone taking 6.001 hates you)
...
**(goodbye)
(goodbye, hal)
(see you next week)
(next!)
(who are you?)
**(eric grimson)
(hello, eric)
(what seems to be the trouble?)
...
**(goodbye)
(goodbye, eric)
(see you next week)
(next!)
(who are you?)
**(suppertime)
(time to go home)
```

Упражнение 6. 4-я стратегия генерации ответов

Реализуйте ещё одну стратегию генерации ответных реплик, зависящую от ключевых слов в реплике пациента. Например, на фразу `i am often depressed` «Доктор» может ответить `when you feel depressed, go out for ice cream`. Структура данных, хранящая группы ключевых слов и привязанных к ним шаблонов для составления ответных реплик должна выглядеть так:

```
(
  ( ; начало данных 1й группы
    (depressed suicide exams university) ; список ключевых слов 1й группы
    ( ; список шаблонов для составления ответных реплик 1й группы
          (when you feel depressed, go out for ice cream)
      (depression is a disease that can be treated)
        )
  ) ; завершение данных 1й группы
  ( ; начало данных 2й группы ...
    (mother father parents brother sister uncle ant grandma grandpa)
    (
          (tell me more about your * , i want to know all about your *)
      (why do you feel that way about your * ?)
        )
  )
  (
    (university scheme lections)
        (
          (your education is important)
          (how many time do you spend to learning ?)
        )
  )
)
```

Можно видеть, что в разные группы ключевых слов могут входить одни и те же слова. Если в варианте ответной реплики есть звёздочка, это значит, что в ответе вместо * «Доктор» укажет ключевое слово, по которому он построил реплику. Например, на фразу пользователя, содержащую слово `father` может быть дан ответ `tell me more about your father , i want to know all about your father`.

Задание упражнения 6. Реализуйте в программе стратегию построения ответов по ключевым словам. Пополните набор групп ключевых слов (не менее чем двумя новыми группами). Пополните варианты ответов для каждой группы (не менее чем двумя вариантами). Построение реплик по ключевым словам должно быть реализовано так, чтобы можно было вносить изменения только в структуру данных, не исправляя код функции, составляющей реплику. Реализуйте случайный выбор ключевого слова для построения реплики, если этих слов несколько во фразе пользователя. Этот выбор должен учитывать количество вхождений слова в реплику пациента. Когда выбор ключевого слова сделан, по нему следует выбрать один из подходящих шаблонов для составления ответной реплики. Предусмотрите учёт ситуации, когда одно и то же ключевое слово относится к разным группам. Для таких слов следует выбирать шаблон из объединённого перечня всех шаблонов, относящихся к каждой группе, куда входит ключевое слово. В выбранном шаблоне следует заменить * на ключевое слово. При этом следует использовать ту реализацию замен, которая уже есть в коде «Доктора». Поскольку построение реплики является обработкой списка, напишите код с использованием уместных функций высшего порядка (`map`, `foldl`, `foldr`, `filter`, `andmap`, `ormap` ...).

Стратегия построения реплики по ключевым словам применима только в том случае, когда в реплике есть какое-то

ключевое слово. Следует описать отдельную функцию-предикат, проверяющую это. Функция-предикат должна быть эффективной. Она не должна делать лишних действий, после того как становится ясно, что стратегия применима. Если для работы функции-предиката не нужна вся структура данных, хранящая группы ключевых слов и привязанных к ним шаблонов, то следует однократно предобработать её, получив необходимые данные, и затем использовать результат однократной предобработки в вызовах функции-предиката.

Блок 3. Упражнение 7. Обобщённый `reply`

Каждый раз при добавлении новой стратегии приходилось переписывать `reply`. Можно переписать его, создав обобщённую версию, которая будет работать с любым подаваемым ему на вход перечнем стратегий. Удобство обобщённого `reply` состоит в том, что при изменении стратегий построения ответных реплик будет достаточно менять сами данные о стратегиях, но не управляющий механизм. Этот универсальный механизм состоит в следующем: во-первых, строится список стратегий, применимых в текущей ситуации; во-вторых, если в построенном списке больше одной стратегии, то выбирается одна из них; в-третьих, выбранная стратегия применяется и её результат будет ответной репликой. На вход механизма подаётся структура данных со сведениями обо всех стратегиях «Доктора». Про каждую стратегию в этой структуре хранится: а) функция-предикат от реплики пользователя, от истории реплик пользователя, от, возможно, других параметров, которая возвращает `#t`, только если стратегия применима, и `#f` -- иначе (Внимание! Должна храниться именно функция!); б) вес стратегии -- натуральное число, которое будет использовано при выборе одной из применимых стратегий (чем больше вес, тем выше вероятность применения стратегии, например, если есть применимые стратегии с весами 2, 3, 1, то первая будет выбрана с вероятностью 1/3, вторая -- с вероятностью 1/2, третья -- с вероятностью 1/6); тело стратегии -- функция, которая по реплике пользователя, по истории реплик пользователя, по, возможно, другим параметрам строит ответную реплику (Внимание! Должна храниться именно функция!). Например, для стратегии "`hedge`" функция-предикат -- это функция всегда возвращающая `#t`; вес этой стратегии = 1 (самый малый); тело этой стратегии -- функция `hedge`.

Выполняя упражнение 7, следует переписать `reply`, добавив ему параметр -- структуру данных о стратегиях построения реплик. Все стратегии ответов, имеющиеся в программе должны быть представлены в этой структуре данных. Каких-либо способов построения ответов вне структуры данных о стратегиях быть не должно. Структура данных о стратегиях не зависит от реплик пользователей и её значение не должно вычисляться более чем один раз. Так как речь снова идёт об обработке списков, пишите код с использованием уместных функций высшего порядка (`map`, `foldl`, `foldr`, `filter`, `andmap`, `ormap` ...). Назначая веса стратегиям, используйте разные значения веса, чтобы взвешенный поиск не превращался в случайный -- `pick-random`. Рекомендуется реализовать аналог `pick-random` -- `pick-random-with-weight`, который получает список элементов, имеющих веса, и выбирает случайный элемент с учётом весов. Рекомендуется перед реализацией «геометрически» решить задачу, т. е. рассмотреть случайный выбор части из составного отрезка (длина части отрезка = весу соответствующего элемента списка).

Предупреждение