Doctor

Общее описание

Данное задание связано с упрощённой версией известной медицинской системы «Доктор», разработанной в Массачусетском Университете в начале 60-х годов профессором Жозефом Вейзенбаумом (Joseph Weizenbaum). Деятельность системы заключается в следующем. «Доктор» ведёт интерактивную беседу с пользователем, имитируя поведение психотерапевта. В приведённом ниже примере диалога, реплики пользователя предварены приглашением `**`, а реакция системы выделена заглавными буквами. Для некоторого упрощения знаки пунктуации опущены, а реплики пользователя и системы представлены в виде списков языка Лисп.

```
==> (visit-doctor 'Bill)
(HELLO, BILL)
(WHAT SEEMS TO BE THE TROUBLE?)

**(everyone hates me)
(WHY DO YOU SAY EVERYONE HATES YOU)

**(whenever i enter a room no one will look me in the eye)
(MANY PEOPLE HAVE THE SAME SORTS OF FEELINGS)

**(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)
(MANY OF MY PATIENTS HAVE TOLD ME THE SAME THING)

**(even my mother hates me)
(YOU FEEL THAT EVEN YOUR MOTHER HATES YOU)

**(even when i was a baby she said i was ugly and stupid)
(PLEASE CONTINUE)

**(she hit me all the time)
(WHY DO YOU BELIEVE SHE HIT YOU ALL THE TIME)
```

Создаётся впечатление, что система понимает и умеет отвечать на реплики пользователя. На самом деле, в основе такого «интеллектуального» поведения системы лежат всего два достаточно простых метода генерации ответов.

Первый метод заключается в замене всех служебных слов, используемых в предложении пользователя от первого лица, например, `I`, `me`, `my`, `am`, на соответствующие служебные слова, но уже от второго лица. Полученный таким образом ответ начинается некоторым общим замечанием, типа `why do you say` или `you seem to think`. Второй метод ещё проще. Жалоба пользователя просто игнорируется, и в качестве ответа система выдаёт что-нибудь такое, `please continue` или `many people have the same sorts of feelings`. Тот или иной метод генерации ответа выбирается с помощью генератора случайных чисел.

Каждая интерактивная программа, включая и сам интерпретатор Лиспа, на самом верхнем уровне состоит из некоторого бесконечного цикла (driver loop). В этом цикле осуществляется ввод данных, их разбор и генерация выходной информации. В примере процедура `visit-doctor` сначала приветствует пользователя-пациента, задаёт первый вопрос и вызывает процедуру `doctor-driver-loop`, содержащую цикл.

```
(define (visit-doctor name)
 (print (list 'hello, name))
 (print '(what seems to be the trouble?))
 (doctor-driver-loop name))
```

В цикле происходит печать приглашения и чтение реплик пользователя. Если пользователь говорит `goodbye`, система выходит из цикла и завершает работу. В противном случае, система порождает ответ в соответствии с той или другой стратегией, описанной выше.

```
(define (doctor-driver-loop name)
 (newline)
 (princ '**)
 (let ((user-response (read)))
   (cond ((equal? user-response '(goodbye))
          (print (list 'goodbye, name))
          (print '(see you next week)))
         (else (print (reply user-response))
               (doctor-driver-loop name)))))

(define (reply user-response)
  (cond ((fifty-fifty)
         (append (qualifier)
                 (change-person user-response)))
        (else (hedge))))
```

Предикат `fifty-fifty`, используемый в процедуре reply, возвращает `true` или `false` с равной степенью вероятности.

```
(define (fifty-fifty)
 (= (random 2) 0))
```

Вводные фразы и замечания общего плана порождаются путём случайного выбора из соответствующего списка.

```
(define (qualifier)
 (pick-random '((you seem to think)
                (you feel that)
                (why do you believe)
                (why do you say))))

(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))))
```

Процедура `replace` представляет каркас процедуры, производящей замену служебных слов в форме от первого лица на соответствующие слова в форме от второго лица. Все вхождения слова `pattern` в предложении `lst` заменяются на слово `replacement`.

```
(define (replace pattern replacement lst)
 (cond ((null? lst) '())
       ((equal? (car lst) pattern)
        (cons replacement
              (replace pattern replacement (cdr lst))))
       (else
(cons (car lst)
      (replace pattern replacement (cdr lst))))))
```

Эта процедура используется в другой процедуре `many-replace`, которая на вход получает реплику пользователя `lst` и список пар для замены `replacement-pairs` вида

```
((<pat1> <rep1>) (<pat2> <rep2>) ... )
```

Все вхождения образца `pat1` в предложении `lst` будут заменены на `rep1`, `pat2` — на `rep2`, и т.д.

```
(define (many-replace replacement-pairs lst)
 (cond ((null? replacement-pairs) lst)
       (else (let ((pat-rep (car replacement-pairs)))
(replace (car pat-rep)
 (cadr pat-rep)
 (many-replace (cdr replacement-pairs)
      lst))))))
```

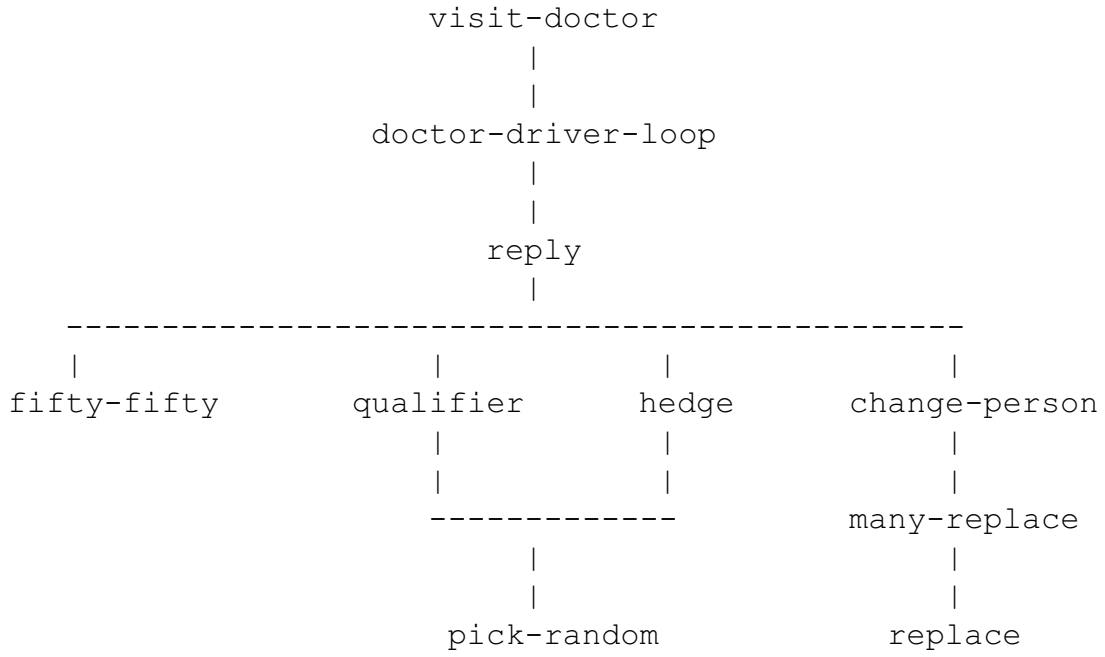
Замена выбранного слова осуществляется с помощью соответствующего вызова процедуры `many-replace`:

```
(define (change-person phrase)
 (many-replace '((i you) (me you) (am are) (my your))
               phrase))
```

Процедура `pick-random`, используемая процедурами `qualifier` и `hedge`, возвращает произвольный элемент списка:

```
(define (pick-random lst)
 (nth (random (length lst)) lst))
```

На следующей схеме представлена последовательность вызовов, осуществляемая в системе `doctor`.



Все перечисленные выше процедуры содержатся в файле `doctor.scm`.

Упражнения

Упражнение 1

Измените процедуры `qualifier` и `hedge`, расширив репертуар системы.

Упражнение 2

Что будет получено в результате вызова процедуры:

```
(change-person '(you are not being very helpful to me))
```

Система `doctor` может быть улучшена, если заменять не только служебные слова в форме от первого лица на соответствующие слова в форме от второго, но и наоборот. Например, если пользователь вводит:

```
(you are not being very helpful to me)
```

система должна ответить что-то типа:

```
(YOU FEEL THAT I AM NOT BEING VERY HELPFUL TO YOU)
```

То есть, `are` должно быть заменено на `am`, `you` на `i`, `our` на `my` и так далее (синтаксическая тонкость замены `you` на VI или `me` может быть проигнорирована).

Реализуйте такую замену с помощью добавления пар

```
(are am), (you i), (your my)
```

к списку образцов в процедуре `change-person`.

```
(change-person '(you are not being very helpful to me))
```

Что возвращает модифицированная процедура? Есть ли зависимость от того, куда поместить новые образцы — в начало или конец списка?

В чем недостаток данной реализации? Напишите корректный вариант процедуры, поддерживающий замену служебных слов обоих типов.

Упражнение 3

Ещё одно улучшение системы состоит в добавлении новой стратегии генерации ответов. Если `doctor` будет запоминать все, что ему говорит пациент, он сможет реагировать следующим образом:

```
(EARLIER YOU SAID THAT EVERYONE HATES YOU)
```

Добавьте реализацию этой стратегии к уже существующей программе, воспользовавшись следующими советами.

Измените программу таким образом, что `doctor-driver-loop` сохраняет список всех реплик пользователя. **Замечание**: не используйте оператор `set!`. В этом нет необходимости.

Процедура `reply`, выбирая эту третью стратегию, берет произвольную реплику пользователя из списка истории, изменяет лицо с первого на второе и добавляет вводную фразу `earlier you said that`.

Для осуществления большего контроля за «случайным» выбором одной из трёх стратегий, вы можете воспользоваться предикатом `prob`:

```
(define (prob n1 n2)
  (< (random n2) n1))
```

Упражнение 4

Текущая версия программы умеет работать только с одним пользователем, чьё имя задаётся в вызове процедуры `visit-doctor`. Когда пользователь говорит `goodbye`, `visit-doctor` возвращает управление интерпретатору Scheme. Измените программу таким образом, чтобы доктор автоматически переходил к приёму следующего пациента после прощания с предыдущим. Предусмотрите некоторый способ завершения работы «многопользовательского» доктора. Например, `visit-doctor` может заканчиваться после того, как определённое количество пациентов обслужено, или введено специальное имя пользователя, например, `suppertime`. Для ввода имени очередного пользователя можете воспользоваться процедурой `ask-patient-name`:

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

Теперь сессия работы доктора может выглядеть следующим образом:

```
==> (visit-doctor)
(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 5

Реализуйте ещё одну возможную стратегию генерации ответа, зависящую от ключевых слов в реплике пользователя. Например, на фразу `I am often depressed` доктор может ответить `When you feel depressed, go out for ice cream`. Список ключевых слов и соответствующих реплик доктора может выглядеть так:

```
 ( ((depressed suicide)
    ((when you feel depressed, go out for ice cream)
     (depression is a disease that can be treated)))
   ((mother father parents)
    ((tell me more about your family)
     (why do you feel that way about your parents?)))
)
```

Более гибкая структура данных может включать зависимость от конкретного ключевого слова из списка. В этом случае на фразу пользователя, содержащую слово `father` доктор будет отвечать `Tell me more about your father`.

```
( ((depressed suicide)
    ((when you feel depressed, go out for ice cream)
     (depression is a disease that can be treated)))
   ((mother father parents)
    ((tell me more about your *)
     (why do you feel that way about your * ?)))
)
```

Упражнение 6

Обобщите структуру программы, используя набор предикатов и ассоциированный с ними список процедур, порождающих ответ. Если реплика пользователя удовлетворяет некоторому предикату, то вызывается одна из соответствующих процедур. Например, имея предикат

```
(lambda (user-response) (< (length user-response) 3))
```

программа может реагировать на короткие реплики пользователя фразой `Could you say more?`.

Упражнение 7

Спроектируйте и реализуйте собственную стратегию генерации ответа, расширяющую способности «доктора».

Список литературы

[1] J. Weizenbaum. Computer Power and Human Reason. 1976.

[2] Sherry Turkle. The Second Self. 1984.