Trabajo Práctico Final
Análisis de Lenguajes de Programación

Bianchi, Gabina Luz

16 de agosto de 2019

Qué es

Este trabajo consiste en el desarrollo de la primera versión del programa Méri. Méri es un identificador de tipos de poesía en castellano, que se basa en la cantidad de versos y los patrones de rima existentes. Actualmente es capaz de identificar: redondilla, cuarteto, seguidilla, romance, soneto y décima. Sin embargo, como se explicará más adelante, se podrían agregar nuevas estructuras poéticas de manera muy sencilla.

Algunos conceptos importantes

A fin de poder entender con mayor profundidad el funcionamiento y desarrollo de este proyecto, se presentan algunas definiciones básicas sobre el dominio específico en cuestión: la poesía.

Rima

Rima es la repetición de una secuencia de fonemas a partir de la sílaba tónica al final de dos o más versos. Se establece a partir de la última vocal acentuada, incluida ésta.
Existen distintas clasificaciones de rimas. Según su timbre, se pueden encontrar dos tipos distintos:

- Rima consonante o perfecta: coinciden todos los fonemas a partir de las vocales tónicas. Por ejemplo:

*selva a su amor, que por el verde su**elo***
*no ha visto al cazador que con desv**elo***

- Rima asonante o imperfecta: coinciden las vocales, pero hay al menos una consonante que no coincide[1]. Por ejemplo:

*Un nombre de mujer, una blanc**u**r**a**,*
*un cuerpo ya sin cara, la pen**u**mbr**a**.*

Patrones de rima

A continuación se presentan las características principales de las estructuras poéticas consideradas en el proyecto. Es importante destacar que, en esta versión, el programa no es capaz de identificar cada una de las propiedades, sino que analiza únicamente cantidad de versos y esquemas de rima, dejando para una extensión futura el estudio de la longitud de los versos.

- Redondilla: estrofa de cuatro versos, normalmente osctasílabos, con patrón de rima consonante *abba*
- Cuarteto[2] estrofa de cuatro versos, normalmente endecasílabos, con patrón de rima consonante *abba*
- Seguidilla: estrofa de arte menor formada por cuatro versos. Los impares, heptasílabos y libres, y los pares, pentasílabos con rima asonante. El patrón de rima es *abcb*.
- Romance: estrofa de arte menor formada por cuatro versos octosílabos, con el primero y el tercero libres, y el segundo y cuarto con rima asonante. El patrón de rima es *abcb*.
- Soneto: es una composición poética compuesta por catorce versos de arte mayor, endecasílabos en su forma clásica. Admite distintos patrones de rima, según la época y el autor. En este caso se considerará uno de los más utilizados: *abbacddceffegg*[3].
- Décima: estrofa constituida por diez versos octosílabos. La estructura de rimas es fija en *abbaaccddc*, pero no hay acuerdo sobre si debe ser asonante o consonante. En Méri se considerará asonante.

[1]En este trabajo se considera a la rima consonante como un caso particular de rima asonante, ya que muchos esquemas de rimas no son estrictos en este sentido.

[2]Por lo mencionado anteriormente, en esta versión de Méri, la redondilla y el cuarteto utilizan la misma métrica, ya que el programa no está preparado para diferenciar la cantidad de sílabas de cada verso.

[3]Basado en libros de sonetos actuales.

Instalación y modo de uso

Para poder utilizar el software primero se deben descargar los archivos fuentes del siguiente repositorio: `https://github.com/gabina/alp`, y tener instalado el sistema de paquetes Cabal[4]. Luego, debe instalarse el programa para poder ejecutarlo, corriendo el siguiente comando en una terminal:

```
$ cabal install
```

Méri tiene dos formas de ser utilizado: de forma interactiva o a través de un archivo. Para iniciar Méri interactivamente, debe utilizarse el siguiente comando:

```
$ cabal run -- -i
```

Una vez iniciado Méri, se debe elegir la opción correspondiente al tipo de poema que se desea verificar, y luego ingresar el poema.

Si se desea trabajar leyendo el poema desde un archivo *poema.txt*, debe correrse el siguiente comando:

```
$ cabal run -- -f poema.txt n
```

n será un entero indicando la estructura poética que se quiere verificar, siguiendo la correspondencia que se presenta a continuación:

```
2 - Cuarteto
3 - Redondilla
4 - Seguidilla
5 - Romance
6 - Soneto
7 - Decima
```

Los poemas a analizar deben ser escritos en una misma línea, separando los versos con una barra (/), e indicando el fin del poema con un asterisco (*). Por ejemplo:

```
soy el que baja al chino de la cuadra/ con lluvia con
burbujas en los charcos/ saludando a porteros medios
parcos/ y al perro del florista que me ladra/*
```

[5]

[4]Cabal es un sistema de paquetes de Haskell que permite a los desarrolladores y usuarios distribuír, usar y reutilizar software fácilmente.

[5]Extracto de un soneto de Pedro Mairal

Méri por adentro

En esta sección se presentarán las ideas principales sobre el diseño y funcionamiento de Méri, detallando cada uno de los módulos involucrados y proveyendo casos de prueba.

Paquete Cabal

Además de los módulos de Haskell específicos para la resolución del problema en sí, un paquete cabal cuenta con dos archivos más. Por un lado, uno con extensión *.cabal*, el cual contiene metadatos sobre el paquete, incluyendo nombre, versión, descripción, tipo de licencia, dependencias, etcétera. Por el otro, un archivo *Setup.hs*. Es un programa de Haskell de un único módulo que realiza tareas de seteo. En este caso se ha dejado con el seteo estándar que se genera automáticamente al crear un paquete cabal con el comando *cabal init*.

Common

En el módulo *Common.hs* se encuentran los tipos de datos utilizados.

Para representar los poemas, los versos y las sílabas, se utilizan sinónimos de tipos:

```
type Verse = String
type Poem = [Verse]
type Syllable = String
```

Para trabajar cómodamente con distintas estrcturas poéticas, se definió un tipo de dato *Metric*, con dos posibles constructores, respondiendo a rima asonante o consonante. Además, este tipo de datos contiene un entero indicando la cantidad de versos correspondiente, y un conjunto de conjuntos de enteros donde se indican qué versos deben rimar entre sí.

```
Metric = Asonante Int (Set (Set Int)) | Consonante Int (Set (Set Int))
```

Por ejemplo, para referirse a una redondilla se debería utilizar:

```
Consonante 4 {{0,3},{1,2}}
```

[6] Se eligió esta tipo de datos con el objetivo de que una métrica dada tenga una única representación posible[7]. Sin embargo, supongamos que se necesita

[6] La notación con corchetes no es válida en Haskell. Es a modo ilustrativo.

[7] Por esta razón se obvió el uso de listas. Las listas imponen un orden en sus elementos que en este caso no sería apropiado, ya que se desea que decir "el verso 0 rima con el 3"sea equivalente a decir "el verso 3 rima con el 0".

representar una estructura poética de 5 versos con patrón de rima asonante *abbab*. En ese caso, lo natural sería utilizar:

```
Asonante 5 {{0,3},{1,2,4}}
```

No obstante, la siguiente sería una representación también válida de la misma estructura poética:

```
Asonante 5 {{0,3},{1,2},{1,4}}
```

Para (ayudar a) evitar la multiplicidad de representaciones, se creó la función *checkMetric*, que se encarga de chequear que todos los conjuntos sean disjuntos uno a uno y que ningún elemento del conjunto sea mayor o igual a la cantidad de versos. En caso de que la métrica no cumpla estas propiedades, el programa falla, lanzando un error interno. Para lograr esto se utiliza la mónada *Input*, presentada a continuación.

```
data Input a = IN { runInput :: IO (Either String a) }
```

La mónada *Input* es, en esencia, una mónada *IO*, que además de tener efectos de entrada/salida, devuelve un valor *Either*.
La mónada *IO* es necesaria ya que el programa debe informar al usuario el resultado: si el poema ingresado corresponde a la estructura poética elegida o no. Además, en caso de que no corresponda, Méri avisa los fallos que encontró.
La mónada *Either String* es muy útil para el manejo de errores. En este caso, los valores correspondientes al constructor *Left* indicarán un error interno, mientras que el constructor *Right* supone el correcto funcionamiento del programa[8].

Functions

En el módulo *Functions.hs* se encuentran todas las funciones definidas para trabajar con poemas y métricas. El algoritmo para separar en sílabas, implementado en la función *syllabifier*, está basado en [2]. Se realizaron modificaciones a la versión original ya que se encontraron algunos errores.

Options

En el módulo *Options.hs* se escriben todos los programas correspondientes a los distintos esquemas poéticos. En esencia, asocia cada tipo de poema a una instancia del tipo *Metric*. Allí es donde se deben agregar nuevos esquemas si fuera el caso deseado.

[8]Esto no implica que el poema ingresado matchee con la estructura buscada, sino que el análisis se está llevando a cabo sin errores.

Parsers

En el módulo *Parsers.hs* se encuentra el parser capaz de consumir el poema ingresado por el usuario.

Main

El módulo *Main.hs* es el módulo principal, desde donde se lanza la interacción con el usuario o se lee el archivo, según corresponda. En caso de agregar un nuevo esquema poético, debería modificarse para agregar la nueva opción al menú.

Casos de prueba

A continuación se presentan casos de prueba exitosos para los distintos esquemas de Méri. También se pueden encontrar en los archivo de la carpeta *Ok*, y pueden correrse utilizando el script *pruebaOk.sh*.

```
Cuarteto

quizá si te quedabas un día más/ o yo fuera más alto más
hermoso/ si fuera un poco menos vergonzoso/ si fuera más
guerrero más audaz/ si yo estuviera al frente de mi vida/
y no en estos cuarteles del invierno/ si me vistiera un
poco más moderno/si hubiera mantenido la embestida/*

Redondilla

por juramento y por arcos hay un bar/ vacío con la mesa
en la que hablamos/ hace casi dos años y hoy estamos /
sin llamarnos sin vernos sin hablar /*

Seguidilla

Pues andáis en las palmas/ ángeles santos/que se duerme mi
niño /tened los ramos/*

Romance

Yo voy como un ciego/ por esos caminos./Siempre pensando en
la penita negra/que llevo conmigo./*
```

```
Soneto

profunda en la ciruela está la casa/ la casa que no está y
una ciruela/ regresa la voz dulce de la abuela/ otra vez un
jardín y una terraza/ el árbol con las ramas acostadas/ la
pileta con sol y sombra y siesta/ la mesa todavía medio
puesta/ las cigarras cayéndose incendiadas/ los grandes se
entregaron al sopor/ quedó el jardín enorme encandilado/
hagan la digestión tengan cuidado/ hermanos aburridos de
calor/ trepados al ciruelo rojo oscuro/ escupen los carozos
del futuro/*

Décima

La vida de entre las manos/se nos escurre VELOZ/Se me atraganta
la voz/de ver como los humanos/perdemos en hechos vanos/el
sentido de la esencia/haciendo que la presencia/en este terreno
hogar/sea un venir, divagar/y perder nuestra existencia/*
```

Nótese que el caso de cuarteto es exitoso habiéndose ingresado 8 versos en lugar de 4. Esto ocurre debido a que ambas estrofas analizadas cumplen las propiedades del cuarteto. Cualquier patrón poético será aceptado siempre que se ingresen todas estrofas que respeten la métrica a verificar.

Los siguientes casos de prueba producen fallas. También se pueden encontrar en los archivo de la carpeta *Error*, y pueden correrse utilizando el script *pruebaError.sh*.

```
Cuarteto - Verso vacío

qué vértigo el abrazo demandante// la pura gravedad que empuja
a adán/ al fondo de la tierra alucinante/*

Redondilla - Cantidad errónea de versos

por juramento y por arcos hay un bar/hace casi dos años y hoy
estamos / sin llamarnos sin vernos sin hablar /*

Seguidilla - Rima faltante

Pues andáis en las palmas/ ángeles santos/que se duerme
```

```
mi niño /tened las mentiras/*
```

Nuevos esquemas

En Méri es muy sencillo incorporar un nuevo esquema de rima, incluso para usuarios sin conocimiento en programación. A continuación se presenta el paso a paso de cómo hacerlo, a través de un ejemplo.

En este caso se agregará el esquema correspondiente a una *sextina*.

1. Tener en claro qué características tiene la estructura poética que se desea incorporar. Si existieran distintas variaciones, se debe optar por una y considerarla como definición. Una sextina está formada por seis versos, endecasílabos u octasílabos, con patrón de rima *aabccb*.

2. Expresar el esquema poético en función del tipo de dato *Metric*. En este caso se utilizará el constructor *Consonante*. Recordar que la primera componente indica la cantidad de versos, y el segundo campo es un conjunto correspondiente a qué versos riman entre sí. En el caso de la sextina, la cantidad de versos es seis. El patrón de rima es *aabccb*, lo cual indica que el verso 0 (primer verso) rima con el verso 1, el verso 2 rima con el último (verso 5), y el verso 3 rima con el verso 4[9].Finalmente la representación quedaría de la siguiente forma:

   ```
   Consonante 6 {{0,1},{2,5},{3,4}}
   ```

3. Escribir la métrica con la notación específica de Haskell, como se muestra a continuación. Aquello escrito en negro es lo que debe quedar fijo, las partes en colores se corresponden con lo pensado en el punto anterior.

   ```
   Consonante 6 (fromList (Prelude.map fromList [[0,1],[2,5],[3,4]]))
   ```

4. Agregar la siguiente función al módulo *Options.hs*, cambiando las palabras en colores, por lo correspondiente según el patrón a incorporar.

   ```
   sextina :: Poem -> Input ()
   sextina p = do f_out "Sextina"
                  m <- checkMetric (Consonante 6 (fromList
                  (Prelude.map fromList [[0,1],[2,5],[3,4]])))
                  showAnswer (satisfyMetric p m)
   ```

[9]Los versos se enumeran comenzando desde el cero. Esto es algo usual en la computación.

5. Incorporar la nueva opción en el módulo *Main.hs*. Para lograrlo se debe agregar un nuevo elemento a *options*, como se indica a continuación.

```
options = [("Ayuda",help),("Salir",bye),("Cuarteto",cuarteto),
("Redondilla",redondilla), ("Seguidilla",seguidilla),
("Romance",romance), ("Soneto",soneto),("Decima",decima),
("Sextina",sextina)]
```

6. Correr el programa como se indica en la sección Instalación y modo de uso.
   El programa debería mostrar la nueva opción en el menú. En caso de que el programa experimente un error interno al comprobar el nuevo patrón, revisar que la métrica haya sido conformada correctamente.

Posibles Extensiones

La principal extensión para realizar en Méri es agregar un contador de sílabas en versos. De este modo, podría controlarse que además de coincidir la cantidad de versos y los patrones de rima en un poema, la longitud de cada verso sea adecuada. Para lograrlo, debería modificarse la función separadora en sílabas, de modo que siga las reglas de separación en sílabas en poesía castellana.

Bibliografía

- Wikipedia
- Heriberto Cuayahuitl.(2004) A Syllabification Algorithm for Spanish.DOI: 10.1007/978-3-540-24630-5_49 [2]
- hackage.haskell.org
- https://www.haskell.org/cabal/users-guide/developing-packages.html