RLisp: Manual de usuario

1 Introducción

RLisp es un paquete Java que permite, en tiempo de ejecución, crear objetos Java y ejecutar sus métodos. Es, por lo tanto, un intérprete de Java. Lo peculiar es que el intérprete es una máquina Lisp completa, por lo que también puede decirse que RLisp es un intérprete Lisp que utiliza la semántica de Java.

2 Licencia

3 Los ficheros

Los ficheros que componen RLisp son cuatro (o tres):

- RLisp.jar es el código ejecutable, y el único imprescindible.
- RLispManS.pdf es este documento que estás leyendo.
  - RLispManE.pdf es la versión inglesa de este documento.
- RLispCode.pdf es un documento que contiene el listado completo del código fuente, y es, por consiguiente, la información definitiva sobre el comportamiento de RLisp.

Para poder acceder desde el programa RLisp, en tiempo de ejecución, a los otros tres documentos, deben colocarse los cuatro ficheros en el mismo directorio. Los ficheros con el código fuente, archivos de extensión java, están dentro de RLisp.jar.

4 La máquina

Para ejecutar RLisp es preciso un sistema informático en el que esté instalada una máquina virtual Java (Java Virtual Machine, JVM).

En concreto, RLisp funciona con la JVM incluida en la versión 1.4.2 del entorno de tiempo de ejecución Java (Java Runtime Environment, JRE) de Sun. Esta máquina se puede descargar de su sitio oficial en Internet. Seguramente funcionará con otras versiones, pero no lo sé.

5 El arranque

Para arrancar RLisp hay que llamar a la máquina virtual Java y decirle que ejecute el código contenido en el fichero RLisp.jar. Eso es todo.

En Windows la orden es:

```
java.exe -jar RLisp.jar
```

6 La ventana principal (ventana amarilla)

Si todo ha ido bien, se abrirá la ventana principal de RLisp, también llamada ventana amarilla. La ventana principal tiene tres partes que, de arriba abajo, son: una barra de herramientas, que veremos en detalle en las secciones siguientes; una parte principal en donde aparecen todas las acciones y reacciones ejercidas, que explicaremos a continuación; y una línea de estado, en donde RLisp escribe algunas informaciones de interés.

En la parte central de la ventana amarilla van apareciendo todas las peticiones realizadas a RLisp y sus respuestas, cada una en una línea. Se distinguen del siguiente modo: las líneas que corresponden a peticiones realizadas a RLisp van encabezadas por el par de caracteres "<<", y las que se refieren a las respuestas efectuadas por RLisp comienzan por ">>". Así, por ejemplo:

```
<< (new RLisp.RPair (string (1 2 3)))
>> (1 2 3)
```

La primera línea indica que se ha pedido a RLisp que construya un nuevo objeto de la clase RLisp.RPair utilizando el constructor RPair(String) y usando como argumento el String de valor (1 2 3), paréntesis incluidos. La segunda es la respuesta de RLisp en la que se muestra el objeto creado. Al resultado de una operación se le puede dar nombre para incorporarlo al diccionario de objetos disponibles, como ya veremos, y esta acción quedaría indicada en la vantana principal así:

```
<< (def lista123 @)
>> lista123
```

Ahora si, por ejemplo, queremos averiguar la clase del objeto creado, la anotación sería:

```
<< (method lista123 'getClass)
>> class RLisp.RPair
```

7 La barra de herramientas (toolbar)

La barra de herammientas tiene cuatro apartados, que llevan nombres ingleses:

- `File` (fichero) permite abrir vías de entrada y de salida de datos, entre ellas los cuadernos de bitácora (log), el árbol (tree) y el teclado (keyboard).
- `Edit` (edición) permite cortar (cut), copiar (copy) y pegar (paste).
- `Action` (acción) permite realizar operaciones de accesibilidad Java (path), permite leer y ejecutar ficheros con código Lisp (Lisp code), y también operaciones relacionadas con el diccionario de objetos (name, unname, list).
- `Help` (ayuda) permite acceder a la documentación.

La estructura completa de la barra de herramientas es como sigue:

- `Toolbar` (barra de herramientas)
  - `File` (fichero)
    - `Load` (cargar)
    - `Load from` (cargar desde)
    - `Save` (salvar)
    - `Save to` (salvar en)
    - —
    - `Tree` (árbol)
    - `Keyboard` (teclado)
    - `Session` (sesión)
    - —
    - `Close` (cerrar)
  - `Edit` (edición)
    - `Cut` (cortar)
    - `Copy` (copiar)
    - `Paste` (pegar)
  - `Action` (acción)
    - `Path` (ruta)
    - `Lisp code` (código Lisp)
    - —
    - `Name` (nombrar)
    - `Unname` (desnombrar)
    - `List` (listar)
  - `Help` (ayuda)
    - `Manual` (manual inglés de usuario)
    - `Spanish Manual` (en español)
    - `Code` (código fuente de `RLisp`)
    - `About RLisp` (sobre `RLisp`)

8 El cuaderno de bitácora (log)

En los barcos se utiliza el cuaderno de bitácora para anotar todos los sucesos significativos que acontecen en él, tales como el rumbo, la velocidad, las maniobras y demás accidentes de la navegación. Para anotar los acontecimientos que suceden al utilizar `RLisp` se usan los ficheros log, ya que *log* es la palábra inglesa para el cuaderno de bitácora. Por convenio, los ficheros log emplean la extensión `log`, aunque el código no lo exige ni lo comprueba, aunque sí lo facilita.

Los cuadernos de bitácora que emplea `RLisp` son ficheros de texto. Esto significa que se pueden crear con cualquier editor de textos, como el `emacs` o el `notepad` de Windows. Si se utiliza un procesador de textos, como Word, hay que tener la precaución de salvar el fichero como texto, es decir, sin códigos de control. Pero la manera más sencilla de crearlos es dejar que lo haga `RLisp`.

Si se pulsa `File-Save to` (salvar en), o sea, si se elige la opción `Save to` después de haber seleccionado `File` en la barra de herramientas, entonces `RLisp` presenta un explorador para seleccionar el nombre y la ubicación dentro del sistema informático de un fichero log. Y, a partir de ese momento, la información que va apareciendo en la consola principal se escribe también en el fichero log.

El cuaderno de bitácora por defecto es el fichero `RLisp.log` del directorio actual. Esto quiere decir que si se ejecuta la orden `File-Save` (salvar), `RLisp` entiende que el cuaderno de bitácora a utilizar es `RLisp.log` del directorio actual, por lo que no es precisa más información.

La información salvada en un fichero log también puede ser utilizada. Si se ejecuta la orden `File-Load from` (cargar desde), `RLisp` presenta un explorador para elegir un fichero. Por defecto sólo aparecen los directorios (también llamados carpetas) y los ficheros (también llamados archivos) de extensión `log`, pero se puede hacer que se muestren todos. Si lo que se quiere es utilizar el fichero `RLisp.log` del directorio actual, entonces se puede ejecutar directamente la orden `File-Load` (cargar). En cualquiera de los dos casos, una vez completada la orden, `RLisp` comienza a procesar el fichero log.

El proceso de un fichero log es sencillo. Si los tres primeros caracteres de una línea no son "`<< `" (el tercero es un espacio), entonces `RLisp` se limita a escribir la línea leída en la consola principal. Si, por el contrario, los tres primeros caracteres de una línea sí que resultan ser "`<< `", entonces, además de escribir la línea completa en la consola, ejecuta el resto de la línea y escribe el resultado en la consola, en una línea que comienza por "`>< `". La información que se escribe en la pantalla durante el proceso de un fichero log no se escribe en ningún fichero log, y esta es la única excepción que tiene la regla.

Cuando se se llama a `RLisp`, se puede añadir el nombre de un fichero log para que se ejecute al comienzo. Por defecto no se ejecuta ninguno.

```
java.exe -jar RLisp.jar filename.log
```

9 El árbol (tree)

Si se ejecuta la orden `File-Tree` (árbol), `RLisp` presenta un explorador para elegir un directorio o un fichero `jar` (Java archive). Y, una vez elegido, nos abre una ventana con el árbol de todas las clases (ficheros con extensión `class`) que son accesibles desde él.

Hay que tener en cuenta que es preciso respetar las reglas de acceso de Java, de modo que si una clase está definida como perteneciente a un `package`, la clase no es accesible desde el directorio en el que se encuentra la clase, sino desde uno jerárquicamente superior. Por ejemplo, si la clase `MyClass` está definida en el

```
package MyApplication;
```

entonces el fichero `MyClass.class` debe encontrarse en un subdirectorio denominado `MyApplication` de cierto directorio, y es desde este cierto directorio desde donde es accesible.

Para cada clase se abren hasta cuatro apartados, que tienen nombres ingleses:

- `Array`, que permite crear un `array` de objetos de esta clase. Este apartado está siempre presente.
- `Fields` (campos), que permite acceder y modificar (si no es `final`) un campo de la clase.
- `Constructors` (constructores), que permite crear un nuevo objeto de esa clase.
- `Methods` (métodos), que permite ejecutar un método de esa clase.

Dentro de cada uno de los cuatro apartados (excepto `Array`) hay varios subapartados. En el apartado `Fields` aparecen todos los campos accesibles de la clase, tanto los de clase (`static`) como los de objeto, y tanto los declarados como los heredados. En el apartado `Constructors` aparecen todos los constructores de la clase, y en el apartado `Methods` todos los métodos.

Una vez seleccionado un subapartado, se apretará el botón `OK` para ejecutarla. Dependiendo de la elección hecha, la aplicación le pedirá datos adicionales. Por ejemplo, si ha elegido un método no estático de cierta clase, le pedirá que seleccione un objeto de los nombrados que pertenezca a dicha clase, y un objeto nombrado de la clase correspondiente para cada uno de los argumentos del método elegido. En todos los casos, en vez de un objeto nombrado, se puede introducir una expresión literal, que `RLisp` evaluará.

La orden ejecutada quedará reflejada en la ventana amarilla a la manera de Lisp, con la petición en una línea que comienza por "`<<`" y el resultado, con la representación textual del objeto Java obtenido, en la línea siguiente, que comienza por "`>>`". Al objeto obtenido se le puede dar nombre apretando el botón `Name` del árbol, o con una orden `Action-Name` de la barra de herramientas, o, como veremos en la sección siguiente, escribiendo la orden `(def name @)` en el teclado, donde `@` es la manera usada en `RLisp` para referirse al último objeto obtenido, y `name` el nombre que queremos darle.

10 El teclado (keyboard)

Al ejecutar la orden `File-Keyboard` se abre una ventana verde que sirve para introducir texto. Así, con el teclado se puede pedir la ejecución de cualquier orden. Estas órdenes han de escribirse a la manera de Lisp. En el apartado sobre Lisp detallaremos qué órdenes entiende concretamente `RLisp`.

La ventana lleva la cuenta de los paréntesis, de modo que si al finalizar una línea no queda ninguno por cerrar, interpreta que la expresión está completa y la pasa a la consola principal para que se ejecute.

Otras maneras de pedir la ejecución desde el teclado hacen uso de los botones de la barra superior: `Nesting`, `Word`, `Maximum`, `Minimum`, `Next` y `Previous`. Todos ellos hacen referencia al lugar en el que se encuentra el cursor. Por ejemplo, al pulsar el botón `Maximum` se ejecuta la expresión completa que incluye el lugar en donde se encuentra el cursor. Si se pulsa `Minimum`, se toma la más pequeña expresión que incluye el lugar en donde se encuentra el cursor. La tecla `Word` pide la ejecución de únicamente la palabra, por lo que, en general, `RLisp` se limitará a responder con su definición en el diccionario. Los botones `Previous` y `Next` se refieren, respectivamente, a la expresión previa y a la posterior. El único botón que no pide una ejecución es `Nesting`, que sirve para que `RLisp` calcule el nivel de anidamiento en el que se encuentra el cursor, y que muestra en la barra de estado que se encuentra en la parte inferior de la ventana verde.

11 La sesión (session)

Si se ejecuta la orden `File-Session`, entonces `RLisp` pasa a aceptar únicamente las órdenes escritas en la consola del sistema. Para volver al modo normal de operación hay que escribir `quit` en la consola del sistema.

12 Finalizar (close)

Al ejecutar la orden `File-Close` se cierran todos los ficheros abiertos que estuvieran manejados por `RLisp`, y se finaliza la ejecución de `RLisp`. Otra manera de finalizar `RLisp` es cerrando la ventana principal. Y, otra forma, es ejecutar desde el teclado la orden:

```
quit
```

Por último, si al leer un cuaderno de bitácora se encuentra una línea:

```
<< quit
```

entonces también se finaliza la ejecución de `RLisp`.

13 Edición (edit)

El apartado de edición (`Edit`) de la barra de herramientas contiene las tradicionales órdenes para cortar (`Cut`), copiar (`Copy`) y pegar (`Paste`), que permiten el paso de información entre aplicaciones.

14 La ruta (path)

La orden `Action-Path` abre un explorador que permite seleccionar un directorio, también llamado carpeta. Una vez seleccionado, el cargador de clases (`ClassLoader`) de Java puede acceder a todas las clases que sean accesibles desde dicho directorio. Caben aquí las mismas observaciones relativas a los `package`s que en el árbol.

Esta acción se puede ejecutar en Lisp, donde `URL` es el directorio, así:
`(path URL)`

15 El código Lisp (Lisp code)

Al ejecutar la orden `Action-Lisp code` (código Lisp), se abre un explorador que permite seleccionar un fichero. En principio sólo muestra los directorios y los ficheros con extensión `lisp`, aunque se puede hacer que muestre todos los ficheros. Se ejecuta cada expresión Lisp del fichero seleccionado.

Esta acción se puede ejecutar en Lisp, donde `URL` es el fichero, así:
`(load URL)`

16 Nombrar (name)

Si se quiere dar un nombre al objeto actual, que es el que aparece escrito en la última línea de la consola principal, hay que utilizar la orden `Action-Name`. Una vez nombrado el objeto, podemos referirnos a él por su nombre, quedando accesible de este modo.

El equivalente Lisp de esta acción, donde `@` es la manera usada en `RLisp` para referirse al último objeto obtenido, y `name` el nombre que queremos darle, es:
`(def name @)`

17 Desnombrar (unname)

Si se quiere que un objeto nombrado y accesible deje de serlo, ha de usarse la orden `Action-Unname`. Se presenta entonces una ventana con una lista desplegable, de donde se puede elegir uno de los nombres actualmente vigentes, para que deje de estarlo.

El equivalente Lisp de esta acción, donde `name` es el nombre desnombrado, es:
`(set! name)`

18 Listar (list)

La orden `Action-List` (listar) muestra todos los objetos actualmente nombrados con su clase Java y su representación textual (`toString()`).

Para obtener la clase de un objeto `object` individual, se puede utilizar la orden Lisp: `(method object getClass)`.

19 Manual de usuario (manual)

La orden `Help-Spanish Manual` (manual) le indica el nombre de este documento: `RLispManS.pdf`. La orden `Help-Manual` le indica el nombre de la versión inglesa de este documento. Si `RLisp` se ejecuta en Windows y el fichero pdf se encuentra en el mismo directorio que `RLisp.jar`, entonces se ejecuta la acción predeterminada sobre él, que suele ser abrirlo con el Acrobat Reader de Adobe.

20 Código fuente (code)

La orden `Help-Code` (código) le indica el nombre del documento que contiene el código fuente del paquete `RLisp`: `RLispCode.pdf`. Si `RLisp` se ejecuta en Windows y el fichero pdf se encuentra en el mismo directorio que `RLisp.jar`, entonces se ejecuta la acción predeterminada sobre él, que suele ser abrirlo con el Acrobat Reader de Adobe.

21 Acerca de RLisp (about RLisp)

La orden `Help-About RLisp` (sobre `RLisp`) muestra la versión de `RLisp` que se está ejecutando, el *copyright* y el nombre de su autor, a sea, mi nombre. No me escriba si no le gusta `RLisp`; si no le gusta, no lo use.

22 Lisp

En esta sección explicaremos las principales características del Lisp utilizado. Obsérvese que se puede utilizar `RLisp` sin tener que escribir una sola línea de Lisp; con el árbol (`tree`) se crean objetos Java, a los que se puede dar nombre con `name`, y, sin salir del árbol, se pueden ejecutar sus métodos y observar sus resultados, a los que también se puede dar nombre. En este modo de funcionamiento, Lisp sólo serve para anotar en la consola principal los acontecimientos que van ocurriendo. Pero con Lisp las posibilidades de `RLisp` crecen enormemente.

22.1 Scheme

Scheme es la variante de Lisp que hemos empleado como guía al desarrollar el Lisp de `RLisp`. Explicaremos las diferencias entre nuestro Lisp y Scheme. Dado que Scheme, es un dialecto bien conocido de Lisp, no lo explicaremos aquí.

22.2 Tail recursivity

La mayor diferencia conceptual con Scheme, es que nuestro Lisp no es *tail recursive*. Y no lo es porque Java (o más concretamente el compilador de Java) no lo es. Si algún día llegara a haber algún compilador Java que fuera *tail recursive*, habría que revisar el código de `RLisp` para conformarlo a los requisitos que impusiese ese compilador.

22.3 Listas impropias

Las listas impropias, o pares, que en Scheme se anotan con un punto (“.”, en inglés *dot*), se anotan en nuestro Lisp con una coma (“,”). La razón de esta diferencia es que

Java utiliza el punto para referirse a objetos, siendo ésta una operación de uso frecuente en `RLisp`. Por ejemplo, para referirse a un método de clase hay que escribir el nombre de la clase, un punto y el nombre del método. Por otra parte, la coma es la manera matemática de expresar un par, `(car,cdr)`.

22.4 Asignación de pares

El asignador de nuestro Lisp es muy potente. La operación primitiva de nuestro Lisp para dar un nombre a un objeto es `def`:

```
(def name object)
```

No evalúa `name`, pero sí `object` y después procede como sigue.

- Si `name` no es un par, o sea, si `(atom? name)` es `t` (de `true`), entonces en el diccionario actual se añade el nombre `name` con el significado `object` (evaluado, como hemos dicho).
- Si `name` es un par y `object` es un par, entonces el `def` se reconvierte en otros dos, obsérvese que `def` es recursiva:
  ```
  (def (car name) (car object))
  (def (cdr name) (cdr object))
  ```
- Por último, si `name` es un par y `object` (evaluado) no es un par, entonces no se produce ninguna asignación.

La primitiva para cambiar el significado de un nombre ya definido es `set!`, y funciona con respecto a los pares del mismo modo que `def`.

```
(set! name object)
```

Si se omite el `object`, `(set! name)`, entonces se desdefine `name`.

22.5 Evaluación de átomos

Si un nombre ha sido definido, y no ha sido desdefinido, entonces se evalúa a su definición. Si un nombre se corresponde con el nombre de una primitiva, entonces se evalúa como tal primitiva. En cualquier otro caso el nombre es nombre de sí mismo, es decir, evalúa al `String` literal que es él mismo (`tres` evalúa al `String` de cuatro letras `tres`, y `12` al `String` `12`), de modo que en nuestro Lisp todos los átomos son evaluables.

22.6 Las primitivas sintácticas: Lisp

`RLisp` es minimalista, y sólo define las siguientes primitivas sintácticas:

- `quote` es como en Scheme (y el apóstrofe "`'`" se usa también como su abreviatura).
- `eval` es como en Scheme.
- `def` y `set!` que ya hemos visto.
- `cond` y `eq?`, donde `cond` es como el de Scheme, excepto que no tiene `else` (úsese `t` en vez), y donde `eq?` es como el `equal?` de Scheme.
- `cons`, `car` y `cdr` que son como los de Scheme, pero `(cons)` es `nil` y `(cons 1)` resulta `(1)`.
- `atom?` es como en Scheme.
- `lambda` que es como en Scheme, pero que usa, en la asignación de formales a actuales, toda la potencia del asignador.

- `rho` que es la manera de crear nuevas formas especiales y extender la sintaxis. Lo explicaremos en detalle a continuación.

22.7 Las extensiones

Para crear nuevas formas especiales y sintaxis, se emplea `rho`. Para evaluar la `rho`-expresión:

```
((rho name expander) expression)
```

primero se evalúa:

```
(expander '(name expression))
```

Y, después, se evalúa el resultado obtenido.

22.8 Las definiciones iniciales

Cuando se arranca `RLisp` se carga automáticamente el fichero `RLisp.lisp`, y otros ficheros Lisp, que veremos después, llamados desde éste. Todos ellos se encuentran dentro de `RLisp.jar`. El fichero `RLisp.lisp` contiene definiciones básicas: `nil`, `t`, `null?`, `not`, `list`, `cadr`, `macro`, `syntax`, `define`, `if`, `sequence`, `or`, `and`, `mapcar`, `let`, `GENV`, `Gdefine`. Un estudio de este fichero puede servir para entender cómo funciona `rho`, ya que en él aparecen muchos ejemplos de su uso (aunque escondidos tras `syntax`). Y una manera muy sencilla de ampliar `RLisp` consiste en añadir definiciones a este fichero.

22.9 Las primitivas semánticas: Java

Además, nuestro Lisp tiene acceso a los objetos Java. Para ello utiliza las siete siguientes primitivas semánticas:

- `(string esto es todo)` produce el `String` de Java `"esto es todo"`.
- `(new cl arg0 arg1 ...)`, en donde `arg` es `ob` o `(cons 'Class ob)`, produce un nuevo objeto de la clase Java `cl` tomando como argumentos los objetos de la lista, teniendo en cuenta lo explicado abajo.
- `(method [ cl | ob ] mt arg0 arg1 ...)`, en donde `arg` es, o bien `ob`, o bien `(cons 'Class ob)`, ejecuta el método `mt` del objeto `ob`, o el `static` de la clase `cl`, con los argumentos de la lista, teniendo en cuenta lo explicado abajo.
- `(field [ cl | ob ] f [ val | ])` devuelve el valor actual del campo `f` del objeto `ob`, o el campo `f` de la clase `cl`. Si aparece `val`, que es opcional, entonces el campo pasa a tener como valor `val`.
- `(array cl ob0 ob1 ...)` crea un `array` de objetos de la clase `cl`, inicializado con los valores `ob0`, `ob1`, etc.
- `(path URL)`, ya vista.
- `(load URL)`, ya vista.

En las primitivas `new` y `method` es preciso indicar la clase de un argumento, utilizando para ello la variante `(cons 'Class ob)`, cuando se quiera utilizar, en vez de la clase a la que pertenece el objeto, una superclase de ésta.

Esto ocurre, por ejemplo, al usar un constructor o un método cuyo argumento es de la clase `java.lang.Object`, como `equals(java.lang.Object)`, ya que aunque todos los

objetos de Java pertenecen a alguna subclase de él, ninguno pertenece esta clase. Así, suponiendo que tanto `object1` como `object2` pertenecieran a una clase que definiera `equals` únicamente de ese modo, para llamarla deberíamos hacer:

```
(method object1 equals (cons 'java.lang.Object object2))
```

Algo parecido ocurre si necesitamos usar el valor `null`, que puede ser el valor de cualquier objeto, pero cuya clase, `void`, no aparece en la signatura de los constructores ni de los métodos.

22.10 Otras definiciones iniciales

El fichero `RLispJava.lisp`, llamado por `RLisp.lisp`, define ocho funciones para obtener valores de los ocho tipos básicos de Java. Así: `(boolean false)`, `(char c)`, `(byte 5)`, `(short 233)` `(int -12)`, `(long 1234567)`, `(float 1.4)`, `(double 4.234567e4)`.

El fichero `RLispMaths.lisp`, llamado por `RLisp.lisp`, define las operaciones aritméticas básicas: suma (`+`), resta (`-`), multiplicación (`*`), cociente (`/`) y resto (`%`); y dos condiciones: igual (`=`) y mayor (`>`). Trabaja con números `java.math.BigInteger`, y la expresión `(# 4)` devuelve el número `BigInteger 4`. Si precisa otro tipo de aritmética ha de redefinir este fichero.

El fichero `RLispArray.lisp`, llamado por `RLisp.lisp`, define las siguientes funciones para trabajar con los `array` de Java: `(isArray? o)`, `(array-length a)`, `(array-get a i)`, `(array-set! a i v)`. Para crear un `array` unidimensional inicializado de la clase `cl` se puede usar `(array cl ob0 ob1 ...)`. Y para crear uno multidimensional no inicializado `(new cl[] dim1 dim2 ...)`.