Universidad Nacional de Rosario
Facultad de Ciencias Exactas,
Ingeniería y Agrimensura
Departamento de Ciencias de la
Computación

Análisis de Lenguajes de Programación

EDSL Keyboard Macro

Cerruti Lautaro

Enero de 2024

Descripción del proyecto

Este trabajo final consiste de un EDSL (lenguaje de dominio especifico embebido en Haskell) para la definición de macros de teclado. Los macros de teclado son muy útiles para la tarea de automatización como pueden ser tareas de testing o scripts para juegos.

Con este proyecto podremos definir un macro utilizando un lenguaje creado específicamente para esto, luego compilar este macro a C, a un ejecutable o incluso correrlo desde el mismo compilador. Se permiten funciones como presionar teclas, tipear lineas enteras, repetir una serie de acciones por tiempo o una cantidad de veces determinada, movimientos de mouse, etc.

Lenguaje y operaciones

La gramática del lenguaje es la siguiente:

```
<prog> ::= <defs> <macro>

<defs> ::= <def> <defs>
  | <empty>

<def> ::= def <ident> = <macro>

<macro> ::= <macro> ; <macro>
  | line "<string>"
  | repeat <int> <macro>
  | timeRepeat <int> <macro>
  | <key> + <macro>
  | sleep <int>
  | rangeSleep <int> <int>
  | usleep <int>
  | rangeUSleep <int> <int>
  | mouse <int> <int>
  | <ident>
  | <key>
  | (<macro>)

<key> ::= <char>
  | <int>
  | <mbutton>
  | <special>

<mbutton> ::= LMB | RMB | MMB
```

```
<special> ::= LARROW | UARROW | DARROW | RARROW
  | SHIFT | LSHIFT | RSHIFT | TAB | RETURN
  | ENTER | ESC | SPACE | CONTROL | LCONTROL
  | RCONTROL | ALT | LALT | RALT | SUPR
  | BACKSPACE | SUPER | LSUPER | RSUPER
  | MENU | F<int>
```

Y el mismo tiene las siguientes operaciones:

- line string: Dado un string, simula el tecleo de la misma.
- repeat n macro: Dado un entero n y un macro, repite el macro la cantidad de veces indicada por el entero.
- timeRepeat n macro: Dado un entero n y un macro, repite el macro los segundos indicados por el entero.
- key+macro: dada una tecla y un macro, ejecuta el macro mientras presiona la tecla.
- sleep n: dado un entero n, realiza un sleep de n segundos.
- rangeSleep m n: dado dos enteros m y n, realiza un sleep de un tiempo aleatorio en segundos en el rango [m,n].
- usleep n: dado un entero, realiza un sleep de n microsegundos.
- rangeUSleep m n: dado dos enteros m y n, realiza un sleep de un tiempo aleatorio en microsegundos en el rango [m,n].
- mouse x y: Dados dos enteros x e y, mueve el cursor a esa posición.
- variable: ejecuta el macro definido previamente en una variable.
- tecla: las teclas comunes como letras o números se escriben sin mas, las teclas especiales se escriben como están definidas en el token 'special' de la gramática.
- LMB, RMB y MMB: son las acciones del mouse, LMB es el click izquierdo, RMB el click derecho y MMB el del medio (el de la rueda del mouse).

Manual de uso e instalación

Este proyecto utiliza Stack para la organización del mismo.

Instalación

Para poder ejecutar el programa primero hay que ejecutar:

```
stack setup
```

y luego compilar el proyecto con:

```
stack build
```

Archivo kb

Para el archivo donde definimos el macro de teclado, la única cosa a tener en cuenta es que primero se definen todas las variables y luego se tiene el macro final a ejecutar como la ultima expresión del archivo. un ejemplo de esto es:

```
def copy = CTRL+C
def paste = CTRL+V
def selectall = CTRL+A

sleep 10; selectall; copy; RARROW; ENTER; ENTER; paste
```

Ejecución

Una vez compilado el proyecto, se puede correr el ejecutable definido en ‘app/Main.hs‘ sobre un archivo ‘.kb‘ haciendo:

```
stack exec EDSL-keyboard-macro-exe -- PATH [-OPT]
```

Las opciones disponibles son:

- ‘-p‘: Imprimir el programa de entrada.
- ‘-a‘: Mostrar el AST del programa de entrada.
- ‘-l‘: Utilizar Linux (X11) como SO.
- ‘-w‘: Utilizar Windows como SO.
- ‘-c‘: Generar código C.
- ‘-e‘: Compilar a un ejecutable.
- ‘-r‘: Correr el Macro.
- ‘-h‘: Imprimir ayuda.

Por ejemplo, para imprimir el programa ‘mouse.kb‘ del directorio ‘test‘, ejecutar:

```
stack exec EDSL-keyboard-macro-exe -- test/mouse.kb -p
```

Para generar el código C y su ejecutable:

```
stack exec EDSL-keyboard-macro-exe -- test/mouse.kb -c -e
```

Tener en cuenta que el código C se puede generar eligiendo la plataforma (con ‘-l‘ o ‘-w‘) y no depende de en que sistema operativo estamos ejecutando el compilador. Esto no es así con la generación del ejecutable o la opción de correr el macro, estas tienen como restricción que el compilador este funcionando en el mismo SO para el que se lo esta tratando de compilar. En caso de no indicar el sistema operativo, por defecto se pondrá el que se este utilizando para correr el compilador. En caso no de indicar si se quiere compilar a C, a ejecutable o correr el macro, por defecto se creara el ejecutable.

Organizacion

Estructura del proyecto:

```
EDSL-keyboard-macro
├── app
│   └── Main.hs
├── src
│   ├── linux_c
│   │   ├── macro_linux.c
│   │   ├── macro_linux.h
│   │   └── macro_linux.o
│   ├── windows_c
│   │   ├── macro_windows.c
│   │   ├── macro_windows.h
│   │   └── macro_windows.o
│   ├── C.hs
│   ├── Common.hs
│   ├── Error.hs
│   ├── MonadKM.hs
│   ├── Parse.y
│   └── PrettyPrinter.hs
└── test
```

En Main.hs tenemos el comportamiento principal del ejecutable del proyecto.

En C.hs se encuentra el modulo encargado de la generación el código C en base al AST del lenguaje.

En Common.hs tenemos el AST del lenguaje junto con la definición del environment utilizado por la monada KM y una función auxiliar para saber si una expresión se encuentra dentro de una lista en base al nombre.

En MonadKM.hs tenemos a la monada utilizada para almacenar las definiciones de variables y manejo de errores.

El archivo Parse.y es la definición del parser del lenguaje y PrettyPrinter.hs es el modulo para la impresión del programa que se toma como entrada.

Por ultimo en las carpetas linux_c y windows_c tenemos una biblioteca que desarrolle para facilitar el código C al que hay que compilar, en estas ya definí las funciones que se encargan de los eventos de mouse y teclado.

Adicionalmente en la carpeta test encontramos una serie de ejemplos de macros.

Diseño

AST

El AST del programa esta definido de la siguiente manera:

```
data Prog = Prog [Def Tm] Tm
```

Donde tenemos dos valores, por un lado la lista de variables definidas y por otro lado el termino que representa el macro a compilar. Siendo los tipos de estos datos los siguientes:

```
data Def i = Def Name i

type Name = String

data Tm =
            Var Name
          | Key Key
          | Usleep Int
          | Sleep Int
          | RangeSleep Int Int
          | RangeUSleep Int Int
          | While Key Tm
          | Repeat Int Tm
          | TimeRepeat Int Tm
          | Line String
          | Seq Tm Tm
          | Mouse Int Int
```

En los términos tenemos todas las posibles operaciones con los datos que estas necesitan para funcionar. En especial podemos notar que cuando se presiona una tecla, ya sea una acción normal o dentro de un while, esta se guarda como un tipo Key.

```
data Key = NKey Char | SKey SpecialKey | MouseButton Int

data SpecialKey = LARROW | UARROW | DARROW | RARROW
          | SHIFT | LSHIFT | RSHIFT | TAB | RETURN
          | ENTER | ESC | SPACE | CONTROL | LCONTROL
          | RCONTROL | ALT | LALT | RALT | SUPR
          | BACKSPACE | SUPER | LSUPER | RSUPER
          | MENU | Fkey Int
```

Las keys pueden ser tres tipos:

- las teclas normales, siendo estas las letras o los números, representadoas por un Char
- los tres botones del mouse, donde a cada uno se le asigna un número del 1 al 3
- las teclas especiales como pueden ser CTRL, ALT, SHIFT, etc

Bibliotecas en C

En las bibliotecas macro_linux.h y macro_windows.h podemos encontrar las funciones utilizadas en la compilación a C, las mismas incluyen las funciones de mover el mouse, presionar teclas o botones del mouse o tipear un string. Las implementaciones de estas funciones se encuentran en macro_linux.c y macro_windows.c.

Es importante destacar que para la compilación de estas bibliotecas se necesitan ciertas bibliotecas especiales presentes en los sistemas operativos correspondientes. Por lo que para compilar macro_linux.c es necesario estar en un Linux que utilice X11 y al momento de compilarlo hacerlo con

```
gcc -c -o macro_linux.o macro_linux.c -lX11 -lXtst -lX11-xcb
```

Y en el caso de Windows es necesario estar en ese SO y compilarlo con

```
gcc -c -o macro_windows.o macro_windows.c -Wno-deprecated-declarations
```

Compilación C

Para la compilación a C tuve que dividir según el sistema operativo que se esta utilizando debido a limitaciones de los mismos.

Para reducir la cantidad de código C generado, las variables se compilan como funciones de C, y luego en el código C generado en el main se llama a estas funciones. Estas variables solo se compilan y se incluyen en el código si son utilizadas por el macro final o por alguna variable que es utilizada en el macro final. Para esto utilizo un método que se encuentra en Main.hs llamada getNeededDefs, la misma recorre el AST del macro final y devuelve una lista de todas las definiciones necesitadas. Esto introduce una restricción, ya que las variables que se llamen de forma recursiva o variables que sean mutuamente recursivas nunca van a poder ser compiladas.

Para este proceso ya mencionado de recorrer las variables y retornar las que se utilizan, es necesario almacenarlas, para eso implemente la monada KM, la que tiene un entorno global para estas definiciones. Sumado a esto también se puede utilizar para generar errores en caso de que una variable no se encuentre y para imprimir por pantalla.

Generación del ejecutable

Para la generación del ejecutable utilizo el método c2exe, este método toma el archivo C generado previamente y lo compila utilizando el gcc. Luego de generar el ejecutable, si la opción de generar el código C no fue utilizada, el archivo c generado es borrado, dejando así solo el ejecutable.

Correr el ejecutable

En caso de querer correr el macro, lo hice fue correr el ejecutable generado en el paso anterior, haciendo uso del método createProcess de System.Process al igual que se había utilizado previamente con el gcc.

Luego de haber ejecutado el macro, en caso de solo haber utilizado la opción de correrlo, el ejecutable creado previamente se elimina.

Limitaciones

Algunas cosas a tener en cuenta es que la funcionalidad de imprimir una linea se desarrollo con la base de una distribución de teclado en ingles estadounidense para decidir sobre que símbolos se utiliza la tecla SHIFT, por lo que es recomendable utilizar esta distribución cuando se utilice algún macro.

Otra limitación es que este proyecto fue desarrollado para Windows y para los Linux que utilicen X11 como manejador de ventana. No funciona en macOS ni en distribuciones de Linux que utilicen Wayland u otro manejador.

Referencias y Bibliografía

[1] *typed-lambda-calculus-interpreter*, LCC 2022, TP3 ALP. URL: https://github.com/Lautaro Cerruti/typed-lambda-calculus-interpreter

[2] *Monadic-LIS-Evaluators*, LCC 2022, TP4 ALP. URL: https://github.com/LautaroCerruti/Monadic-LIS-Evaluators

[3] *Compilador, LCC 2023*. URL: https://github.com/LautaroCerruti/compiladores2023

[4] *SendInput Documentation*. URL: https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-sendinput

[5] *Virtual-Key Codes*. URL: https://learn.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes

[6] *XTest Documentation*. URL: https://www.x.org/releases/X11R7.5/doc/Xext/xtestlib.html