Integración de CLIPS en C y Python

Lukas Häring García

¿Cómo se instala CLIPSPy?

Se realizará la instalación en el sistema operativo Windows, para ello, vamos a instalar Anaconda 3. Este contiene la distribución de Python y una IDE, además de tener el instalador de paquetes “pip”. En nuestra consola de comandos utilizaremos:

```
> pip install clipspy
```

¿Se pueden evaluar en Python expresiones de CLIPS? ¿Cómo?

```
-- Importamos
import clips

-- Creamos el enterno
entorno = clips.Environment()

-- Creamos una cadena larga, ya que las expresiones pueden ser muy largas
clips_cadena = """
   <Expresión en lenguaje CLIPS>
"""
-- Evaluamos la cadena
entorno.eval(clips_cadena)
```

Mediante CLIPSPy se pueden utilizar dentro de CLISP funciones de python. ¿Cómo se hace?

```
import clips
environment = Environment()

def python_funcion(argumentos):
    <Contenido de la función en python>

-- Creamos el entorno
entorno = clips.Environment()

-- Asociamos la función de python con una de Clips
entorno.define_function(python_funcion)

-- Podemos probarla utilizando la función descrita anteriormente
```

```
entorno.eval('(python_funcion <argumentos>)')
```

¿Qué métodos de CLIPSPy asertan y retractan un hecho en CLISP?

Cómo podemos esperar, se trata de una traducción de dichos nombres:

```
import clips
environment = Environment()

-- Para Asertar
hecho = entorno.assert_string(“<hecho en cadena formato CLIPS>”)

-- Para Retractar
hecho.retract()
```

¿Cómo se ejecutaría en Python un sistema basado en reglas definido mediante un fichero .clp?

```
import clips
environment = Environment()

-- Cargamos un archivo “clp”
environment.load(“nombre_archivo.clp”)

-- Borramos el entorno previo
environment.reset();

-- Ejecutamos todas las reglas
environment.run()
```

Describe brevemente cómo convertirías un sistema basado en reglas definido mediante un fichero .clp en un fichero ejecutable

Como hemos comentado anteriormente, estamos utilizando windows, por lo que haremos uso de MinGW como compilador.

1. Inicia CLIPS y carga todo lo que va a constituir el módulo ejecutable. LLamaremos la regla **constructor-to-c** de la siguiente manera.

```
CLIPS> (constructor-to-c <nombre archivo> <id> [<ruta destino> [<elm-maximos>]])
```

   El id se trata de un entero, “elm-maximos” limitador para el tamaño del ejecutable.

2. En el archivo “setup.h” obtenido, modificamos la variable “RUN_TIME” a valor de 1.

3. Compilamos todos los archivos obtenidos.

Describe brevemente cómo incluirías en CLISP una función definida en C

Para ello vamos a modificar “userfunctions.c”, en archivo, vamos a implementar la función que deseemos.

```
<valor devuelto> <nombre> (<argumentos){
    … Código
    return <Resultado>;
}
```

Finalmente dentro de la función “EnvUserFunctions”, añadimos el declarador del puntero y la definición de clips.

```
void EnvUserFunctions(void *entorno) {
   -- Puntero
   extern double <nombre funcion>(void *);
   -- Definición en clips
   EnvDefineFunction(
       entorno,
       "<nombre>",
       '<valor devuelto>', -- Primera letra (Por ejemplo double -> ‘d’)
       PTIEF <nombre>,
       "<nombre>"
    );
}
```

Describe brevemente cómo incluirías un sistema basado en reglas definido mediante un fichero .clp dentro de tu programa escrito en C

De manera similar a Python, utilizando la variable de entorno:

```
int main(){ -- Aunque puede hacerse en otra función.
   …
   entorno = CreateEnvironment();
   …
   EnvLoad(entorno, “nombre archivo.clp”);
   EnvReset(entorno); -- Borramos el entorno previo
   EnvRun(entorno, -1); -- Ejecutamos el archivo clp al completo
   ...
}
```

¿Qué funciones se utilizan para asertar o retractar un hecho en un sistema basado en reglas embebido en un programa de C?

Para asertar una nueva regla, se puede realizar de dos formas, pero la más cómoda:

```
(void*) hecho = EnvAssertString(entorno, “hecho”); -- Puntero a dicho hecho
```

La otra forma, es utilizar “EnvAssert” y crear a mano el hecho en C.

Para retractar una regla, de forma equivalente a Python:

```
EnvRetract(entorno,  hecho); -- Dónde hecho es de tipo (void*)  (Puntero)
```

¿Se pueden ejecutar varios sistemas basados en reglas distintos dentro de un mismo programa de C?

Sí, además, estos se ejecutan de forma concurrente, para crearlos de manera equivalente a la definición de 1 solo:
Se puede utilizar “main.c” o “envrnmnt.h” para la declaración de estos:

```
int main(){
   void* entorno1 = CreateEnvironment();
   void* entorno2 = CreateEnvironment();
   … Código para cada entorno
   Por ejemplo:
   EnvLoad(entorno1, “clips_e1.clp”);
   EnvLoad(entorno2, “clips_e2.clp”);
   ....
   EnvReset(entorno1);
   EnvReset(entorno2);
   EnvRun(entorno1);
   EnvRun(entorno2); -- Ejecutados de forma concurrente.
   DestroyEnvironment(entorno1);
   DestroyEnvironment(entorno2);
}
```