Ejercicio Guiado 3

Desarrollo de Software

Raúl Aguilar Arroyo, 100472050

Carlos Seguí Cabrera, 100472060

Grupo 3

27 mar 2023

ÍNDICE

Funcionalidad #1

Clases de equivalencia y análisis de valores límite

En primera instancia hemos tenido que identificar las clases de equivalencia, primero una clase válida para cada posible valor correcto: de product_id siendo un EAN13 correcto,; order_type premium o regular; address de la forma → (ASCII()∅ASCII ()...){>1}{20-100}; phone_number → 0-9 {9}; zip_code válido en España → 0-9 {5}.

Del mismo modo probamos cada caso con un valor inválido siendo por ejemplo para product_id → non existent type y de esta forma formamos el resto de clases inválidas.

En cuanto a los valores límite de cada entrada de product_id es 13; address es 1 de espacios en blanco, 2 de cadenas, 20 y 100 (rango de las direcciones); phone_number es 9 ; zip_code es 5. Por ello debemos comprobar estos valores límites sumando y restando en diferentes casos como el mismo valor límite.

Con ello formamos las diferentes clases de equivalencia siendo en nuestro caso 17 pruebas ya que en un caso se puede comprobar más de 1 si esta es correcta como puede ser nuestro primer ejemplo en el que al ser todos los valores correctos comprobamos valores correctos de cada entrada. Tenemos por ello 17 test para 22 clases.

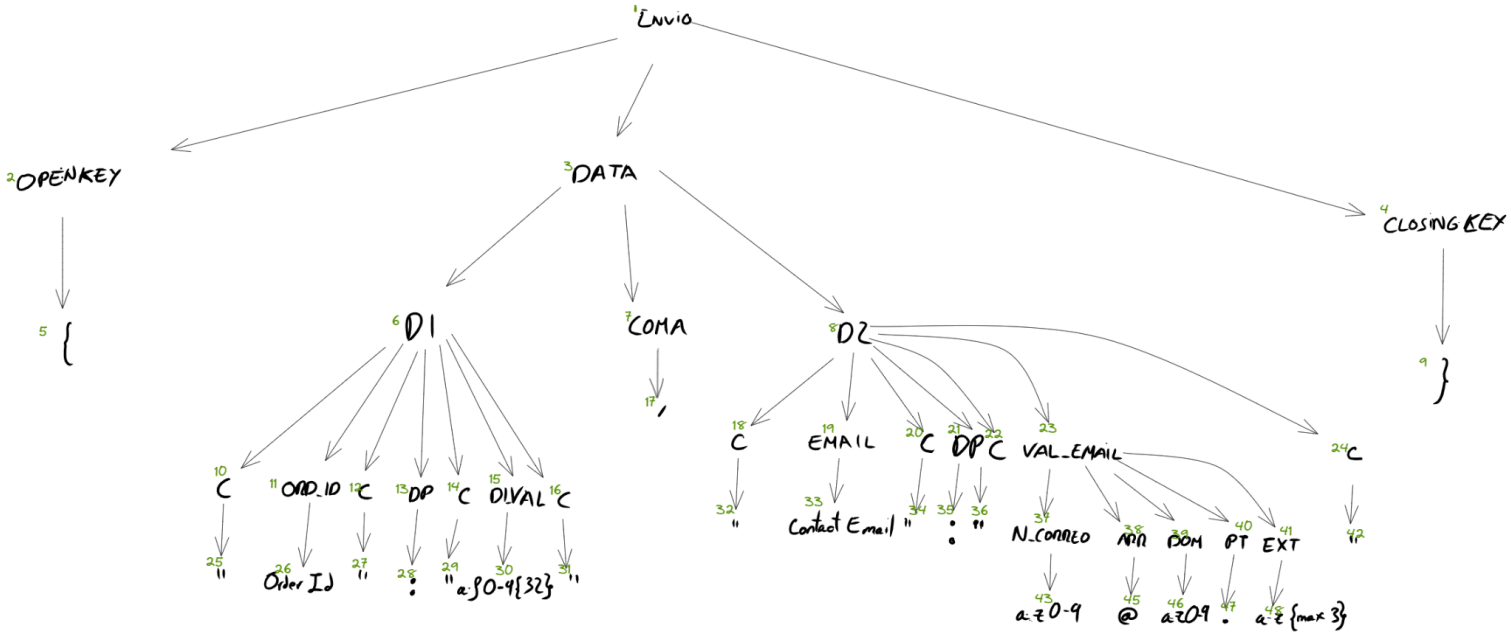
Para comprobar las salidas lo implementamos en función de las entradas cubriendo todos los casos posibles: tanto MD5 CODE, como el fichero con los datos del pedido y todas las posibles excepciones.

Funcionalidad #2

Gramática

```
Unset
ENVIO:= OPENKEY DATA CLOSINGKEY
OPENKEY:= {
CLOSINGKEY:= }
DATA:= D1 COMA D2
D1:= C ORD_ID C DP C DI_VAL C
C:= "
ORD_ID:= OrderId
DP:= :
DI_VAL:= a..f0..9{32}
COMA:= ,
D2:=C EMAIL C DP C VAL_EMAIL
EMAIL:= ContactEmail
VAL_EMAIL:= N_CORREO ARR DOM PT EXT
N_CORREO:= a..z0..9
ARR:= @
DOM a..z0..9
PT:= .
EXT:= a..z{max 3}
```

Árbol de derivación



Selección de casos de prueba (análisis sintáctico)

Tras realizar nuestro árbol de derivación tenemos 47 nodos, de los cuales hay 21 terminales y 26 no terminales.

Para ello identificamos en el caso en el que las entradas son válidas, por ello en primera instancia tenemos nuestro caso válido, en el cual todos los nodos terminales tienen valores correctos.

Para los casos inválidos tenemos dos posibilidades:

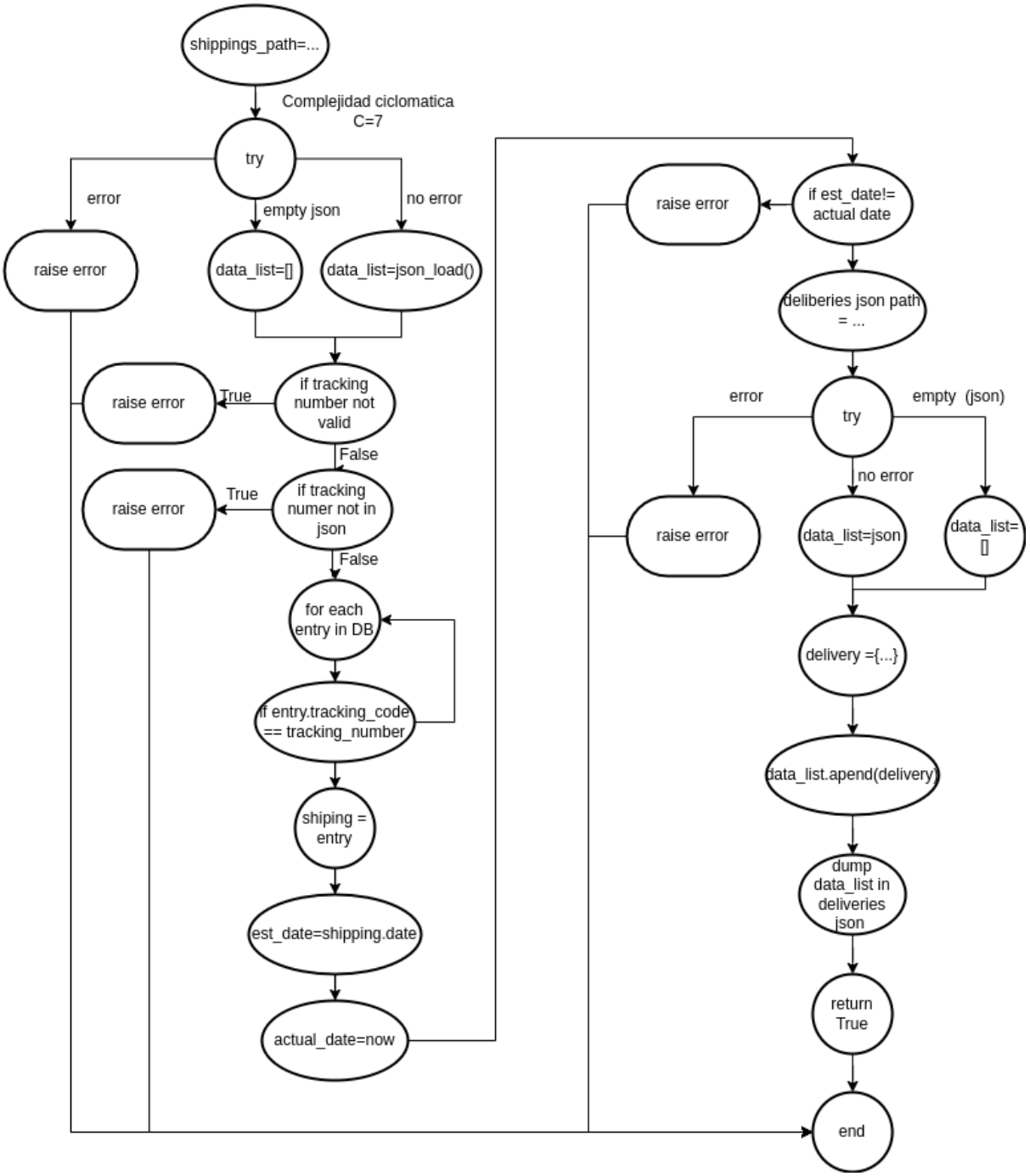
- Nodos no terminales → Se eliminan y se duplican los nodos
- Nodos terminales → modificar valores inválidos

Debido a que no tenemos en cuenta el primer nodo que es el conjunto, tendríamos 72 casos de prueba (NO TERMINALES 2 *+TERMINALES +CASO VÁLIDO* = 25*2 +21+1).

Con ello ya disponemos de todos los casos de prueba(detallados en el excel). Para ello generamos los json con una función json_generator y realizamos los test con la clase MyTestCase en la cual definimos el caso válido y los terminales y no terminales.

Funcionalidad #3

Diagrama de flujo de control



Rutas básicas

En esta función hemos tenido que utilizar el análisis estructural. Para ello hacemos hacemos nuestro diagrama de flujo y calculamos la complejidad ciclomática que en este caso es 7, con lo cual elegimos un camino básico y de ahí vamos cambiando decisiones de una en una, teniendo el total de los caminos siguientes:

1. [1,2,5]
2. [1,2,4,6,7]
3. [1,2,3,6,8,9]
4. [1,2,3,6,8,10,11,10,11,12,13,14,15,16]
5. [1,2,3,6,8,10,11,12,13,14,15,16,17,18,23]
6. [1,2,3,6,8,10,11,12,13,14,15,16,17,18,19,22]
7. [1,2,3,6,8,10,11,12,13,14,15,16,17,18,20,22]

Nota: no se ha probado el caso en el que no existen los json.

Casos adicionales (Bucles)

No hemos tenido que realizar casos de prueba adicionales con el bucle for, ya que este se prueba en las entradas base.