# Descripción del sistema

La Figura 1 muestra el diagrama de la arquitectura de nuestro sistema. A continuación, describimos someramente sus elementos para posteriormente profundizar en los mismos. La entrada del sistema es un conjunto de programas Python obtenidos tanto de GitHub como de una asignatura de primer año de programación de un Grado en Ingeniería Informática del Software de la Universidad de Oviedo (Capítulo 4.1), para utilizar código escrito por principiantes y expertos. La salida es una colección de las categorías sintácticas más frecuentemente utilizados, su relación con el nivel de experiencia del programador y un informe acerca de los patrones atípicos detectados.

Diagrama

Descripción generada automáticamente

Figura 1: Arquitectura del sistema propuesto para la extracción de información sintáctica.

Lo primero que hacemos es modificar el compilador de Python de la Python Standard Library (PSL), aumentando la información sintáctica ofrecida por éste. Para ello, rediseñamos su AST, añadiendo nuevos nodos y relaciones, proporcionando así una información mucho más detallada que la ofrecida por el compilador original (Capítulo 3.1).

Establecemos una clasificación de los distintos tipos de subárboles con estructura común, Identificamos los siguientes 7 tipos de nodos: proyecto (conjunto de directorios y ficheros Python que componen un proyecto independiente), módulos (fichero Python), definiciones de clases, definición de funciones, definiciones de métodos, sentencias y expresiones.

Para algunos de estos 7 tipos principales vamos a definir tablas auxiliares para almacenar cierta información extra. En el caso de los módulos, definiremos los imports. Para complementar la tabla de definiciones de funciones, definiremos la tabla de parámetros (conjunto de parámetros con los que se define una función) qué, también ayudarán para expandir la información acerca de las Lambda expresiones. Como tablas derivadas de sentencias vamos a definir 2, la tabla de cases (información extra relativa a las sentencias de tipo Match) y la tabla de handlers (información extra relativa a las sentencias Try y TryStar). Por último, para complementar las expresiones, vamos a definir 5 tablas: comprehensions (información relativa a generadores de listas, diccionarios, tuplas y sets), invocaciones a funciones, cadenas de texto formateadas, variables, vectores (información relativa a los literales de tipo lista, diccionario, tuplas y sets).

(Capítulo 3.3).

Posteriormente, los datos de las tablas son procesados y filtrados para posteriormente proceder con la detección de anomalías. Si las anomalías son debidas a errores de medición (entradas que no debían haber sido consideradas) se eliminan; en caso contrario, se incluyen en un informe de anomalías.

## Modificación del compilador de Python

El primer paso consiste en implementar una versión modificada del compilador de Python para obtener información sintáctica con un mayor nivel de detalle de los nodos de los AST generados por el compilador de Python. Hemos utilizado el compilador de la Python Standard Library ast.

Para cada fichero Python, el compilador realiza un análisis sintáctico del mismo y genera el AST original. Una vez creado el AST, se recorre para sustituirlo por otro con información más detallada sobre la estructura sintáctica del programa. Recorremos el AST original mediante el patrón de diseño *Visitor* [20].

A modo de ejemplo, todas las operaciones binarias en Python son representadas en el PSL como instancias de BinOp, dificultando el conocimiento del tipo concreto de operación que representa. Para dar más información acerca de la expresión, hemos añadido nodos que representan operaciones binarias concretas en función del operador: operadores simples (Arithmetic), potencias (Pow), shifts (Shift) o a nivel de bits (BWLogical), entre otras (Capítulo 3.2). De este modo, se obtiene una versión con información más detallada de las construcciones sintácticas utilizadas por el programador.

## Nueva estructura del AST

El diseño del AST de la Python Standard Library ast está formado por un total de 56 clases [22]. Nosotros añadimos 9 nuevas clases y campos, tal y como se muestra en la Figura 2, para facilitar el análisis de los datos.

Nuestro diseño identifica nuevas estructuras y las trata de forma independiente como, por ejemplo, las definiciones de funciones y clases de forma independiente a las sentencias. También especializa varias clases pertenecientes a las construcciones sintácticas de sentencia y expresión. A modo de ejemplo, la Python Standard Library modela cualquier operación binaria utilizada mediante una instancia de su clase BinOp. Esta clase no distingue los diferentes tipos de operaciones como, por ejemplo, no distingue “x + y” de “0 xor 1”. Esta información extra nos permite identificar que la primera suele ser más habitual entre los programadores noveles, pudiéndose detectar como patrón común a este tipo de programadores. La segunda es más comúnmente utilizada por programadores expertos. Por ello, nuestro diseño del AST incluye los siguientes tipos de nodos para las operaciones binarias: Arithmetic, Pow, Shift, BWLogical, MatMult, UnaryAritmetic, UnaryNot, UnaryBWNot.

Lo mismo sucede cuando declaramos un valor o un literal. La Python Standard Library interpreta todos los literales como instancias de una misma entidad Constant. Nuestro sistema puede, de forma relativamente sencilla, conocer su tipo y añadir mayor especificación. Definimos las siguientes clases para clasificar una expresión literal: IntLiteral, FloatLiteral, ComplexLiteral, NoneLiteral, BoolLiteral, StringLiteral, EllipsisLiteral.

Además de las nuevas clases, nuestro diseño también añade nuevos campos para almacenar información sintáctica de interés. Por ejemplo, nuestros nodos del AST incorporan información relativa al rol que éstos juegan en su construcción sintáctica padre. Así, una asignación puede jugar dos roles distintos si su padre es una sentencia while: ser la condición del while o una sentencia de su cuerpo. La información que denota cuándo una asignación se utiliza como condición es significativa, por ejemplo, para detectar programadores no principiantes (éstos rara vez usan asignaciones en las condiciones de while). Estas modificaciones nos permiten ampliar el nivel de detalle en comparación con la versión original, donde solamente conoceríamos la clase que define a un nodo y las que definen a sus hijos.