Inferencia y verificación de propiedades en sistemas ciber-físicos estocásticos

Título en inglés: Inference and verification of properties in stochastic cyber-physical systems

Director: José Ignacio Requeno

Autores: Javier Romero Flores (GIC) & Dmytro Vernyuk (GII)

Trabajo de fin de grado del Grado en Ingeniería de Computadores e Ingeniería Informática

Facultad de Informática

Universidad Complutense de Madrid

Curso 2021-2022

I ain't no physicist but I know what matters.

– Popeye el Marino

Resumen (español)

Blablabla

Palabras Clave: Blablabla

Abstract (English)

Blablabla

keywords: Blablabla

Prefacio

Prefacio.

Índice general

Capítulo 1

Introducción

1.1. Sistemas ciberfísicos

Los sistemas ciberfísicos son un tipo especial de sistemas en tiempo real que combinan un micro controlador o programa software, representado mediante una máquina de estados discretos, con uno o varios sensores que interactúan sobre una variable física. Ejemplos de este tipo de entornos son los ordenadores de a bordo que regulan la velocidad de crucero de un vehículo, o el piloto automático que controla la altitud y trayectoria de un avión.

Garantizar el correcto funcionamiento de estas plataformas es crucial, ya que un mal funcionamiento conlleva una reducción del comfort por parte de los usuario o, incluso, la pérdida de vidas humanas. Una manera de asegurarlo es mediante el uso de técnicas de verificación formal. Las técnicas de verificación en tiempo de ejecución [**STTT_RV_21**] (*runtime verification* en inglés) implementan un monitor que supervisa que la ejecución actual cumple con los requisitos especificados. Los monitores actúan como guardianes, alertando al usuario cuando la ejecución se desvía del comportamiento deseado e, incluso, implementando contramedidas para revertir ese hecho y redirigir el sistema hacia un estado saludable.

Existen diversos formalismos para definir los comportamientos deseados y no deseados. Los requisitos se pueden expresar operacionalmente, mediante algún tipo de autómata de estados finito, o declarativamente, mediante una descripción lógico-matemática. Dentro de esta segunda categoría, la lógica temporal es un tipo de lógica modal que expresa propiedades sobre un *estado* en particular del sistema, o sobre los *caminos* (es decir, secuencia de estados) que atraviesa. En este trabajo utilizaremos Signal Temporal Logic [**STL**], un tipo de lógica temporal enfocada al análisis de señales analógicas o reales.

1.2. Lógica Temporal

Si navegamos por internet o buscamos en los libros acerca de qué es la lógica temporal tarde o temprano daremos con una cita del filósofo alemán Kant que cita:

"la posibilidad de principios apodícticos de las relaciones del tiempo, o axiomas del tiempo en general. Tiene una sola dimensión: los distintos tiempos no son simultáneos, sino sucesivos"

Y, aunque quizás sacado de todo contexto, es un buen primer comienzo para explicar acerca de qué es el tiempo y por consiguiente qué tiene que ver la lógica con éste. Y es que Kant quería expresar que el tiempo es algo que va ocurriendo constantemente, es un conjunto de hechos que juntos transcurren en una dirección y conforman el tiempo, pues bien, teniendo en cuenta esta idea podemos llegar a comprender y diferenciar la lógica temporal.

La lógica proposicional no tiene la persepción de tiempo, en este tipo de lógica tenemos hechos que son verdaderos o falsos, que unidos pueden transformarse en verdad o no, pero nunca individualmente cambian su valor, por ejemplo si decimos que:

s = "El día está soleado"

Este argumento será eterno individualmente, aún asi realicemos operaciones con él como negarlo $\neg s$, todo el argumento se verá afectado en todo momento:

$\neg s$ = "El día no está soleado"

Pero como todos podemos percibir, en la realidad esto no sucede, en otras palabras, puede que "no todo el día esté soleado", quizás si por la mañana y no por la tarde o visceversa, esto es lo que trata de expresar la lógica temporal. En un momento determinado una expresión puede ser verdad mientras que en otra hora, minuto, segundo, o cuálquier otra unidad que utilicemos para medir el tiempo , puede cambiar y tener un valor diferente.

1.3. Objetivos

Los objetivos de este proyecto son:

- extender las capacidades de la lógica temporal STL para expresar propiedades que involucren *tendencias* (derivadas), o *acumulaciones* (integrales),
- implementar esos nuevos operadores lógicos en las herramientas software actuales; y, por último,
- proporcionar una interfaz de usuario amigable que facilite la interacción con dichas herramientas.

En particular, los objetivos de proyecto se han plasmado en contribuciones concretas sobre las siguientes herramientas software preexistentes:

- **STLEval** [**StlEval**]: Es una herramienta capaz de procesar especificaciones en STL y evaluarlas sobre una señal real. Ha recibido la implementación de los nuevos operadores lógico temporales que calculan derivadas e integrales sobre una señal.

- **ParetoLib** [**ParetoLib**]: Es una librería de de minería que aprende las configuraciones (in)válidas del sistema monitorizado, expresadas como plantillas o especificaciones paramétricas en STL. ParetoLib internamente se conecta con STLEval para la ejecución de los cálculos. Ha recibido una actualización de los binarios y librerías DLL de STLEval que empaqueta de forma conjunta con el resto de la librería, así como la interfaz gráfica de usuario. La nueva interfaz gráfica de usuario permite interaccionar tanto con la librería de aprendizaje como indirectamente con la herramienta STLEval.

1.4. Organización del documento

El documento está dividido en 4 capítulos. En el capítulo 3 detallaremos la semántica e implementación de los nuevos operadores lógicos. El capítulo 4 está dedicado a la nueva interfaz gráfica de usuario. Continuamos con las conclusiones más relevantes del proyecto en el capítulo 5. Finalmente el capítulo X incluye ideas para el trabajo futuro. Los anexos X recogen imágenes, código adicional, ...

1.5. Planificación temporal y esfuerzo

Este proyecto tuvo una duración de 8 meses y ha sido realizado por dos estudiantes a tiempo parcial, Dmytro Vernyuk (GII) y Javier Romero Flores (GIC), con una dedicación media de 3 horas al día. Los alumnos han estado tutorados por José Ignacio Requeno mediante reuniones semanales. Se han modificado o creado aproximadamente 1300 líneas de código, que están repartidas entre las dos herramientas software actualizadas y la nueva interfaz gráfica de usuario. El código desarrollado en este proyecto puede consultarse en ramas dedicadas a las nuevas funcionalidades de los correspondientes repositorios web (rama *derivative* de STLEval y rama *GUI* de ParetoLib).

El proyecto arrancó el 13 de septiembre del 2021, ha durado 8 meses y ha constado de 4 fases:

1.5.1. Primera - Investigación

El proyecto se inició con una primera fase de investigación, donde nos dedicamos a adquirir los conocimientos teóricos necesarios para el trabajo: leer información sobre qué es la lógica temporal y la principal diferencia con otras lógicas modales. Recurrimos tanto al material proporcionado por el instructor como al material externo disponible para aprender las principales definiciones y sintaxis básicas.

Ésta fue la parte más complicada de todo el trabajo no sólo por la dificultad teórica de la información especializada en el ámbito de la verificación, sino también por el cambio de paradigma que supone comprender el concepto principal de ésta lógica: que una afirmación se puede convertir en negación dependiendo del instante en el que se evalúa la expresión.

1.5.2. Segunda - Documentación

Después de adquirir los conocimientos teóricos previos necesarios para la comprensión de las funcionalidades a implementar, continuamos con la lectura de la documentación de las librerías y herramientas software que debíamos extender: su arquitectura software y jerarquía de clases. Dedicamos gran parte del esfuerzo a estudiar la definición de los operadores de derivación e integración propuestos en la literatura científica, y su posterior codificación e incorporación a la herramienta software STLEval.

En esta fase, nuestra mayor dificultad fue 1) la labor de aprendizaje de la estructura interna del código de STLEval, y 2) la adquisición y aplicación de las nociones de derivación e integración sobre señales temporales.

1.5.3. Tercera - Implementación

Implementamos las operaciones de derivación e integración de señales temporales en la herramienta software STLEval. Completamos esta tarea con un conjunto de pruebas sobre señales temporales con características *benignas* que nos facilitaron ejecutar tests de validación: por ejemplo, la integral de una señal periódica (sinusoidal o triangular) simétrica sobre el eje horizontal debe ser cero en determinados intervalos.

1.5.4. Cuarta - Diseño y Finalización

Como última tarea implementamos la interfaz gráfica de la aplicación, partiendo de un primer boceto que estabilizamos al cabo de dos reuniones. La tarea más complicada de este proceso fue la conexión entre los diferentes componentes de la interfaz con los métodos propios de las herramientas STLEval y ParetoLib, que ejecutan toda la lógica del programa.

La conexión con STLEval se realizó mediante una API preexistente en C++ que ya

estaba integrada en Python a través de la librería ParetoLib. Este hecho, la posibilidad de acceder tanto a la librería de minería ParetoLib como indirectamente a la herramienta STLEval, motivó la elección de Python como lenguaje de programación para el diseño de las ventanas de usuario. La facilidad de uso del lenguaje así como la cantidad de librerías auxiliares para el tratamiento de datos con las que extender las funcionalidades de la interfaz en el futuro contribuyeron también a esta decisión.

1.6. Organización de Equipo

Debido a la dificultad teórica de este proyecto, decidimos realizar las tareas de manera conjunta: todos los apartados de este proyecto a excepción de la labor de documentación se realizaron a la par por ambos integrantes del equipo.

La planificación coincide con las fases del proyecto, indicamos las fechas exactas en el siguiente gráfico:

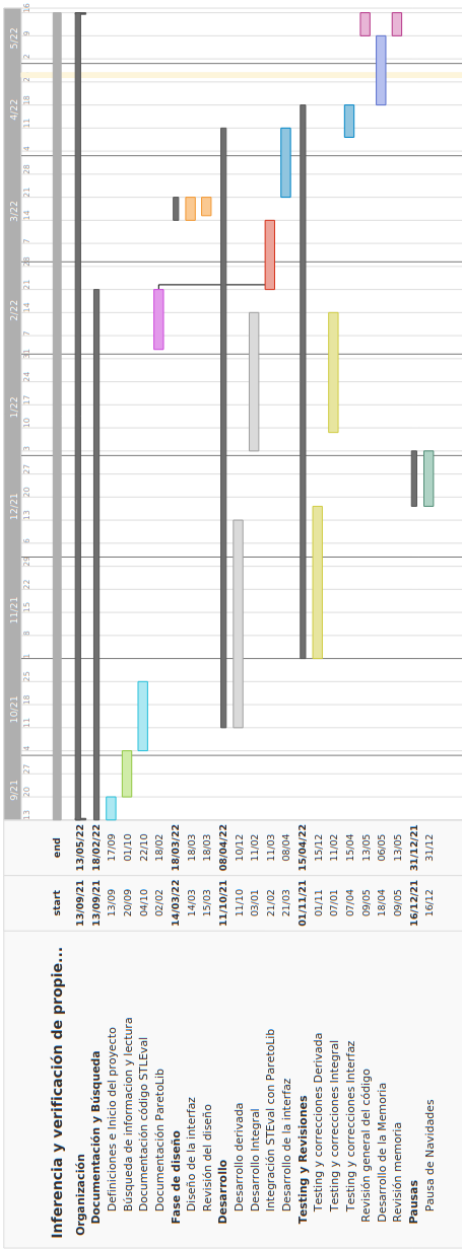

Figura 1.1: Diagrama de Gantt.

Capítulo 2

Introduction

2.1. Cyber-physical system

Cyber-physical systems are a special type of real-time systems that combine a microcontroller or software program, represented by a discrete state machine, with one or more sensors that interact with a physical variable. Examples of this type of environment are the on-board computers that regulate the speed of a vehicle, or the autopilot that controls the altitude and trajectory of an aircraft.

Ensuring the proper functioning of these platforms is crucial, since a malfunction leads to a reduction in user comfort or even the loss of human lives. One way to ensure this is through the use of formal verification techniques. Runtime verification techniques [**STTT_RV_21**] implement a monitor that monitors that the current execution meets specified requirements. Monitors act as keepers, alerting the user when execution deviates from desired behavior and even implementing countermeasures to reverse that fact and redirect the system back to a healthy state.

There are various formalisms to define desired and undesired behaviors. Functional requirements can be expressed operationally, through some kind of finite state automaton, or declaratively, through a logical-mathematical description. Within this second category, temporal logic is a type of modal logic that expresses properties about a particular *state* of the system, or about the *paths* (ie, sequence of states) that it traverses. In this work we will use Signal Temporal Logic [**STL**], a type of temporal logic focused on the analysis of analog signals.

2.2. Objectives

The objectives of this project are:

- Extend the capabilities of temporal logic to express properties involving *trends* (derivatives), or *accumulations* (integrales),
- Implement these new logical operators in current software tools, and
- Provide a friendly user interface that facilitates interaction with said tools.

In particular, the project objectives have been translated into concrete contributions to the following pre-existing software tools:

- **STLEval** [**StlEval**]: It is a tool capable of processing STL specifications and evaluating them on a real signal. It has received the implementation of the new time-logic operators that calculate derivatives and integrals on a signal.
- **STLEval** [**StlEval**]: It is a mining library that learns the (in)valid configurations of the monitored system expressed as templates or parametric specifications in STL. ParetoLib internally interfaces with STLEval for the execution of the calculations. It has received an update of the STLEval binaries and DLL libraries which package together with the rest of the library, as well as the graphical user interface. The new GUI allows to interact both with the learning library and indirectly with the STLEval tool.

2.3. Document Organization

The document is divided into X chapters. In the chapter 3 we will detail the semantics and implementation of the new logical operators. The 4 chapter is dedicated to the new graphical user interface. We continue with the most relevant conclusions of the project in the chapter 5. Finally ....

2.4. Time schedule and effort

This project lasted 8 months and was carried out by two part-time students, Dmytro Vernyuk (GII) and Javier Romero Flores (GIC), with an average dedication of 3 hours per day. The students have been tutored by José Ignacio Requeno through weekly meetings. Approximately 1300 lines of code have been modified or created, which are distributed between the two updated software tools and the new graphical user interface. graphical user interface. The code developed in this project can be consulted in the new functionalities

of the corresponding web repositories (*derivative* branch of STLEval and *GUI* branch of ParetoLib).

The project started on September 13, 2021, and has consisted of 4 phases:

2.4.1. First - Research

The project started with a first phase of research, where we dedicated ourselves to theoretical knowledge necessary for the work: reading information about what is temporal logic and the main difference with other modal logics. We resorted both to the material provided by the instructor and to the external material available to learn the main definitions and basic syntax. This was the most complicated part of the whole work not only because of the theoretical difficulty of the specialized information in the field of but also because of the paradigm shift involved in understanding the main concept of this logic: that an assertion can become a negation depending on the instant at which the expression is evaluated.

2.4.2. Second - Documentation

After acquiring the previous theoretical knowledge necessary for the understanding of the functionalities to be implemented, we continued with the reading of the documentation of the libraries and software tools to be extended: their software architecture and class hierarchy. We devoted a great deal of effort to study the definition of the derivation and integration operators proposed in the scientific literature, and their subsequent coding and incorporation into the STLEval software tool.

2.4.3. Third - Implementation

We implemented the operations of derivation and integration of time signals in the STLEval software tool. We completed this task with a set of tests on time signals with *benign* characteristics that facilitated us to run validation tests: for example, the integral of a periodic signal (sinusoidal or triangular) symmetric about the horizontal axis should be zero in certain intervals.

2.4.4. Fourth - Design and Completion

As a last task we implemented the graphical interface of the application, starting from a first sketch that we stabilized after two meetings. The most complicated task of this process was the connection between the different components of the interface with the methods of the STLEval and ParetoLib tools, which execute all the program logic.

The connection with STLEval was made through a pre-existing API in C++ that was already integrated in Python through the ParetoLib library. This fact, the possibility of

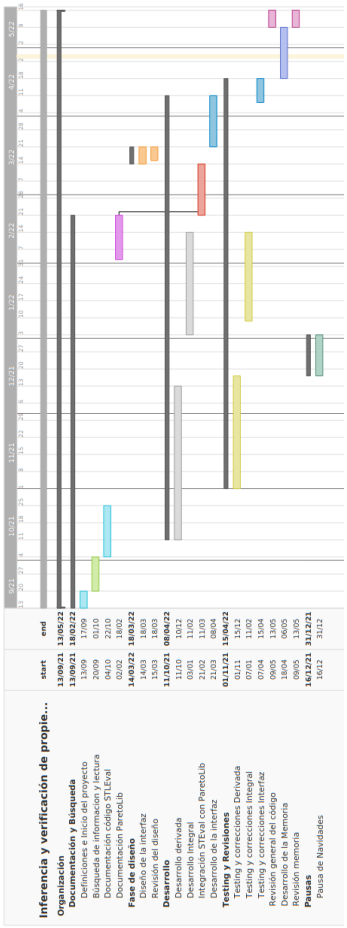

Figura 2.1: Diagrama de Gantt.

accessing both the ParetoLib mining library and the STLEval tool indirectly, motivated the choice of Python as the programming language for the design of the user windows. The ease of use of the language as well as the number of auxiliary libraries for data processing with which to extend the functionalities of the interface in the future also contributed to this decision.

2.5. Team Organization

Due to the theoretical difficulty of this project, we decided to carry out the tasks jointly: all the sections of this project, except for the documentation work, were carried out at the same time by both members of the team. The planning coincides with the phases of the project, the exact dates are shown in the following chart:

Capítulo 3

Nuevos operadores para STL

3.1. Signal Temporal Logic

Signal Temporal Logic (STL) [**STL**] es un tipo de lógica temporal especializada en el análisis de señales reales analógicas. STL permite expresar características sobre la evolución de algún atributo físico, como la velocidad o temperatura. Por tanto, como punto de partida, STL requiere una muestra o traza de ejecución sobre la que comprobar las hipótesis.

La lógica distingue dos tipos de situaciones: propiedades que se satisfacen en un *estado* o momento puntual de la traza de ejecución, o propiedades de *camino* que se evalúan a lo largo de una secuencia de eventos. STL proporciona operadores para recorrer los diferentes estados del sistema y comprobar en qué momentos se cumplen las propiedades.

Por ejemplo, la proposición atómica $v > 120$ mostraría los instantes en los que la velocidad supera los 120. Los operadores de camino enriquecerían esa expresión para analizar si la velocidad se sobrepasa puntualmente en algún momento (**F**)uturo del viaje, (**G**)eneralmente a lo largo de todo el recorrido, o permanece constante hasta (**U**ntil) que se incrementa a un nuevo valor.

Formalmente, la gramática básica de STL comprende los siguientes operadores:

$$\varphi \ := \ \mu \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 U_{[t_1,t_2]} \varphi_2$$

Donde μ representa las proposiciones atómicas, en forma de desigualdades sobre la señal muestreada (p.ej., $\mu := v > 120$); y φ las especificaciones sobre los caminos. El resto de los operadores se componen a partir de los operadores precedentes, donde $t_1, t_2 \in \mathbb{R}_{\geq 0}$ son

marcas temporales que definen el intervalo de monitorización respecto al instante inicial, siendo $t_2 \geq t_1$.

$$F_{[a,b]}\varphi \equiv \top U_{[a,b]}\varphi \qquad\qquad G_{[a,b]}\varphi \equiv \neg F_{[a,b]}\neg\varphi$$

Existen versiones extendidas que permiten analizar aspectos cuantitativas con STL, por ejemplo, los valores máximos/mínimos [**TACAS_19**] o integrales en un intervalo, o calcular la derivada en un punto [**Stl_Der_Int**].

Las especificaciones en STL se *compilan* en un monitor que supervisa la ejecución del sistema (p. ej., el sensor de velocidad de un vehículo). Algunos intérpretes de STL son AMT [**AMT2**] (sintaxis básica) y STLEval [**StlEval**] (sintáxis básica y operadores de min/max). En este proyecto, implementaremos en STLEval los operadores cuantitativos de STL que permiten calcular integrales y derivadas sobre una señal, según la definición propuesta en [**Stl_Der_Int**].

3.2. STLEval

STLEval es una herramienta capaz de manejar tanto expresiones STL básicas, que devuelven señales Booleanas, como ciertas extensiones cuantitativas, que transforman la señal de entrada en una nueva señal continua (operadores de min/max). Por estos motivos, así como su eficiencia (está escrita en en C++) STLEval se ha tomado como punto de partida para implementar los operadores cuantitativos de STL que permiten calcular integrales y derivadas sobre una señal, según la definición propuesta en [**Stl_Der_Int**].

Internamente, STLEval representa una señal como una serie temporal, es decir, una sucesión de pares *(clave, valor)* donde la *clave* es una marca temporal y el *valor* representa la magnitud física en ese momento. La imagen 3.1 ilustra la señal original y su representación interna en STLEval.

3.3. Definición de los nuevos operadores

El cálculo de la derivada e integral se aproximan mediante las siguientes expresiones matemáticas, donde $\mathbf{x}$ representa la señal completa, $\mathbf{x}_\tau$ es el valor de la señal en el instante τ, $\mu^i_{[a,b]}$ es la integral en el intervalo $[a,b]$ y μ^d_+ (μ^d_-) representan la aproximación de la derivada por la derecha (izquierda) del instante temporal en cuestión:

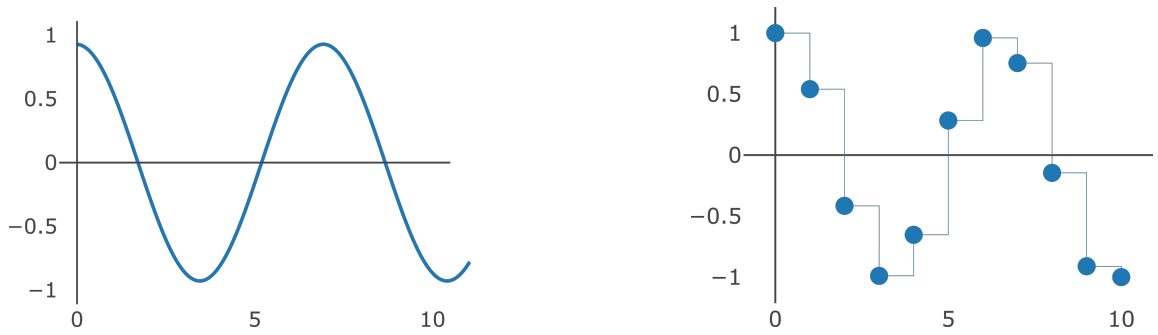

Figura 3.1: Señal original y su reconstrucción.

$$\mu^d_+ = \frac{dg(\mathbf{x})}{dt^+} \geq c \qquad \mu^i_{[a,b]} = \int_a^b g(\mathbf{x}_\tau)\delta\tau \geq c$$
$$\mu^d_- = \frac{dg(\mathbf{x})}{dt^-} \geq c$$

Dado que internamente STLEval representa las señales reales como una serie numérica discreta, aproximamos el instante temporal τ como $\tau = k\delta t$ donde δt es la frecuencia de muestreo y k el término de la sucesión:

$$\mu^d_+ = g(\mathbf{x}_{(k+1)\delta t}) - g(\mathbf{x}_{k\delta t}) \geq c\delta t \qquad \mu^i_{[a,b]} = \sum_{k'=k+a/\delta t}^{k+b/\delta t-1} g(\mathbf{x}_{k'\delta t})\delta t \geq c$$
$$\mu^d_- = g(\mathbf{x}_{k\delta t}) - g(\mathbf{x}_{(k-1)\delta t}) \geq c\delta t$$

Gráficamente, la derivada se interpreta como la pendiente entre dos puntos consecutivos de la señal; y la integral como el sumatorio del área de los rectángulos con base δt contenidos en el intervalo (Figura 3.2).

Los nuevos operadores se han incorporado a STLEval. En la notación de la herramienta, D simboliza el operador derivada ($D \equiv \mu^d_+$), e I representa el operador integral ($I_[a,b] \equiv \mu^i_{[a,b]}$).

3.4. Integración con ParetoLib

ParetoLib [**FORMATS_19**, **ParetoLib**] es una librería de minería que recibe una especificación paramétrica o plantilla en STL y devuelve el rango de valores de las variables

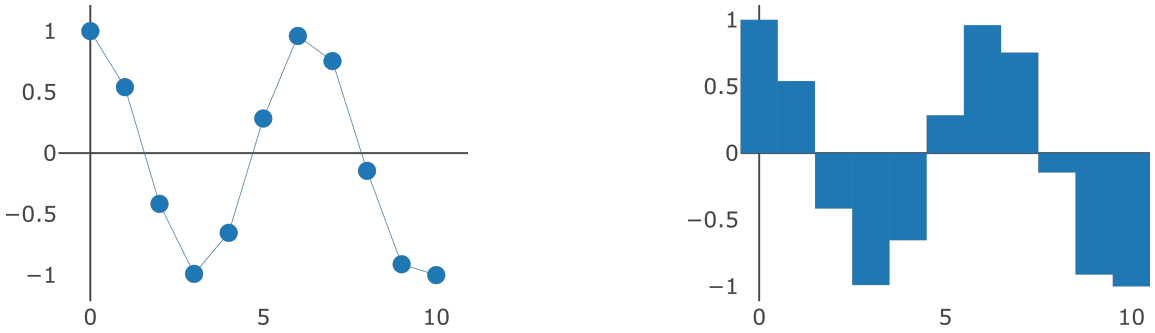

Figura 3.2: Interpretación de la derivada e integral.

para las que la propiedad se satisface o invalida. Internamente, ParetoLib implementa un algoritmo de búsqueda que guía el aprendizaje y evalúa instancias concretas de la fórmula temporal a través de la herramienta STLEval. Para ello, ParetoLib se empaqueta junto los binarios y librerías dinámicas de STLEval precompilados.

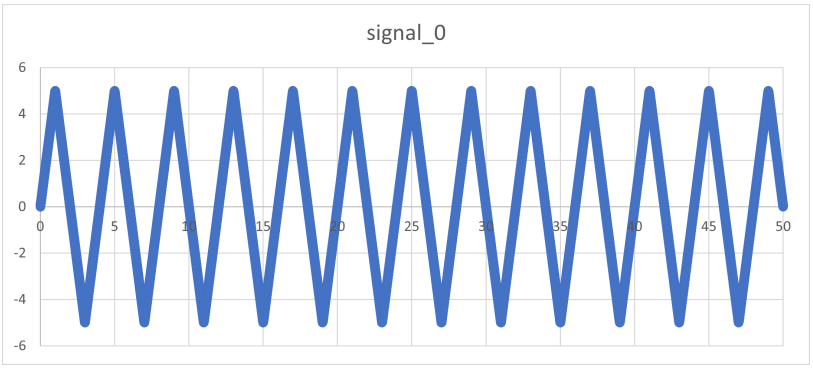

Figura 3.3: Señal triangular

Gracias a la actualización de STLEval, podemos evaluar expresiones que involucren parámetros en derivadas e integrales. Por ejemplo, dada una señal triangular (Figura 3.3) evaluamos sobre ella la siguiente especificación STL[1]:

$$G_{[0,p_1]} D\,\mathbf{x}_0 >= 6 - p_2$$

donde p_1 y p_2 son los parámetros, $\mathbf{x}_0$ es la señal y D es el operador derivada. Como resultado, ParetoLib devuelve la imagen 3.4 que muestra las configuraciones de los parámetros p_1 y p_2 que satisfacen la propiedad (región en verde) y las que lo falsifican (región en rojo).

[1] La ecuación muestra los operadores con notación infija por legibilidad, aunque STLEval realmente utiliza notación prefija. Es decir, se escribiría $(G_{[0,p_1]}(>= (D\,\mathbf{x}_0)6 - p_2))$

Cabe mencionar que este tipo de mapas ya estaban previamente disponibles como resultado de la ejecución de la librería de minería.

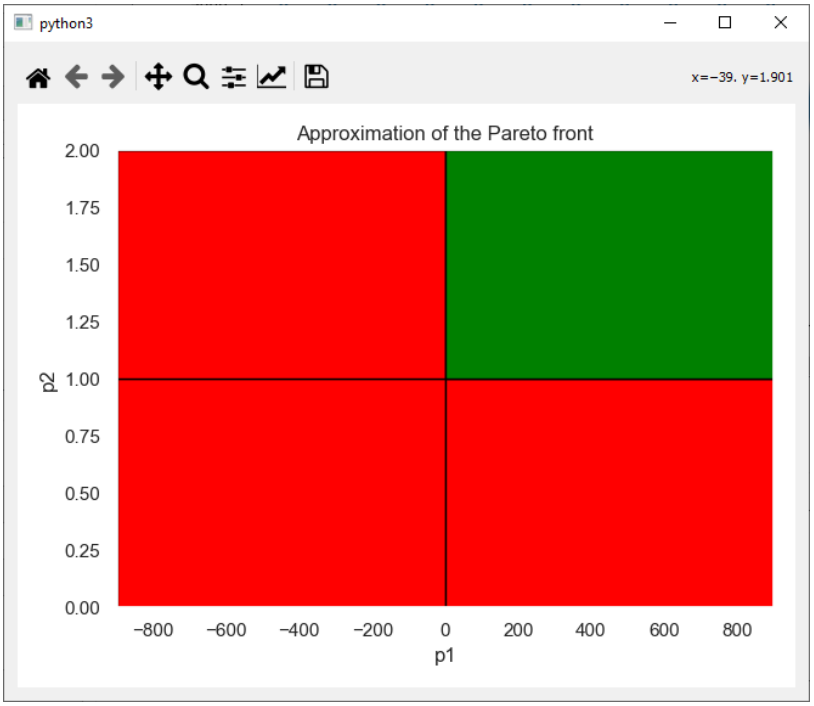

Figura 3.4: Resultado paramétrico

ParetoLib está escrito en Python. El código fuente 3.4 ilustra la forma tradicional de invocar a la librería de minería. En el capitulo 4 veremos la nueva interfaz gráfica que va a facilitar al usuario el trabajo con ParetoLib y con STLEval.

```
1  from ParetoLib.Oracle.OracleSTLe import OracleSTLeLib
2  from ParetoLib.Search.Search import Search2D, EPS, DELTA, STEPS
3
4  # File containing the definition of the Oracle
5  nfile = '../../../Tests/Oracle/OracleSTLe/2D/triangular/derivative/
       triangular.txt'
6  human_readable = True
7
8  # Definition of the n-dimensional space
9  min_x, min_y = (1.0, 0.0)
10 max_x, max_y = (999.0, 2.0)
11
12 oracle = OracleSTLeLib()
13 oracle.from_file(nfile, human_readable)
14 rs = Search2D(ora=oracle,
15                min_cornerx=min_x,
```

```
16                  min_cornery=min_y,
17                  max_cornerx=max_x,
18                  max_cornery=max_y,
19                  epsilon=EPS,
20                  delta=DELTA,
21                  max_step=STEPS,
22                  blocking=False,
23                  sleep=0,
24                  opt_level=0,
25                  parallel=False,
26                  logging=True,
27                  simplify=True)
```

Capítulo 4

Interfaz gráfica

Hasta ahora, las herramientas de verificación y minería mostradas en este proyecto utilizaban interfaces de usuario poco amigables para usuarios inexpertos: a través de línea de comandos para la herramienta STLEval; o mediante código fuente en Python para la librería ParetoLib.

En este apartado proponemos un espacio donde el usuario tenga que simplemente adjuntar los fichero de datos y operaciones (o elegir la operación a realizar desde las opciones disponibles) para finalmente poder visualizar y descargar la salida de señales en forma gráfica.

4.1. Librerías utilizadas

Para el desarrollo de la parte gráfica de este proyecto hemos hecho uso de las librerías:

- PyQt
- Matplotlib
- Pandas
- Seaborn

4.1.1. PyQt

Esta librería está basada en la biblioteca gráfica QT y nos servirá para realizar el diseño de la aplicación. En nuestro caso optamos por un diseño simple de 2 columnas: En la parte izquierda tendremos los botones y opciones para importar los ficheros de señales y la especificación de la operación, además de un espacio adicional para otras operaciones nuevas a

implementar en un futuro. Al lado derecho tendremos la visualización de las señales tratadas y la especificación STL importada.

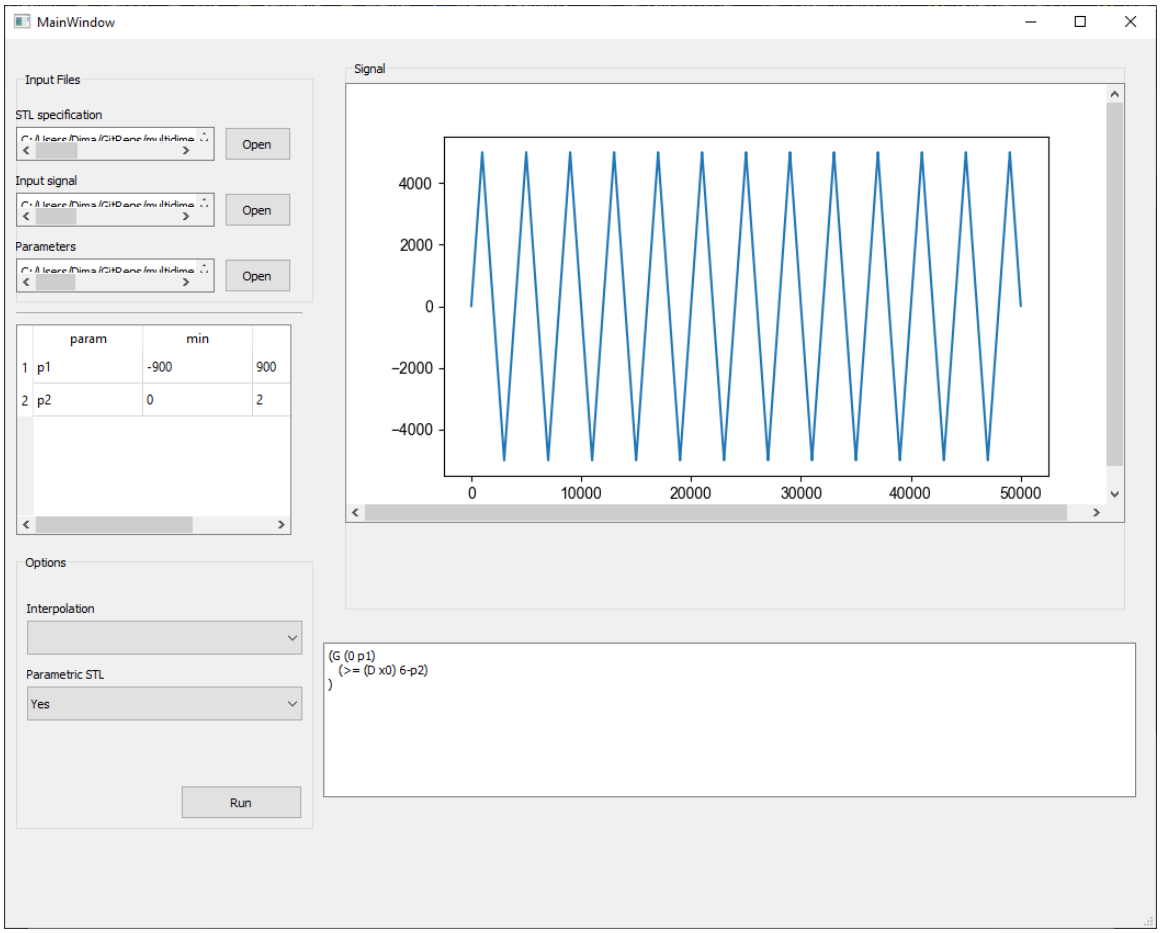

Figura 4.1: La ventana principal

4.1.2. MatplotLib

Hacemos uso de esta biblioteca para la generación de gráficos. En nuestro caso, los datos extraidos para su reproducción son almacenados en un array o mapa el cúal a su vez obtiene los datos de un fichero csv que previamente es tratado (con la librería Pandas) con el fin de obtener el formato adecuado para tu uso.

Los gráficos generados muestran las señales resultantes de la aplicación del conjunto de operaciones especificadas por nuestros ficheros STL. Decidimos mostrar las señales en forma gráfica porque creemos que es una manera bastante fácil de poder comprender y comparar

resultados. Añadido a esto podemos descargar las imágenes en formato png para su posterior uso.

4.1.3. Pandas

Librería de Python especializada en el manejo y análisis de estructuras de datos. En nuestro caso usada para la lectura de los ficheros csv, los cuales contienen los datos de entrada de las señales.

Con esta librería realizamos la transformación y almacenamiento de nuestras señales en una estructura de datos tipo "MAP", el cúal posteriormente utilizaremos tanto para la realización de las operaciones pasándolo como parámetro, como también para la visualización de los datos en forma gráfica haciendo uso de la librería MatplotLib mencionada anteriormente.

4.1.4. Seaborn

Basada en MatplotLib, es una librería que permite generar fácilmente elegantes gráficos. Nosotros la utilizamos para dibujar señales de los datos leídos de los archivos cvs.

La función de esta librería es meramente estética, creemos que este recurso es principalmente un añadido atractivo a la herramienta que puede ser util en el caso que se quiera realizar presentaciones sobre sus resultados gráficos; además damos pie y esperamos poder realizar una mejora en todas las partes más visuales del aplicativo.

4.2. Estructura

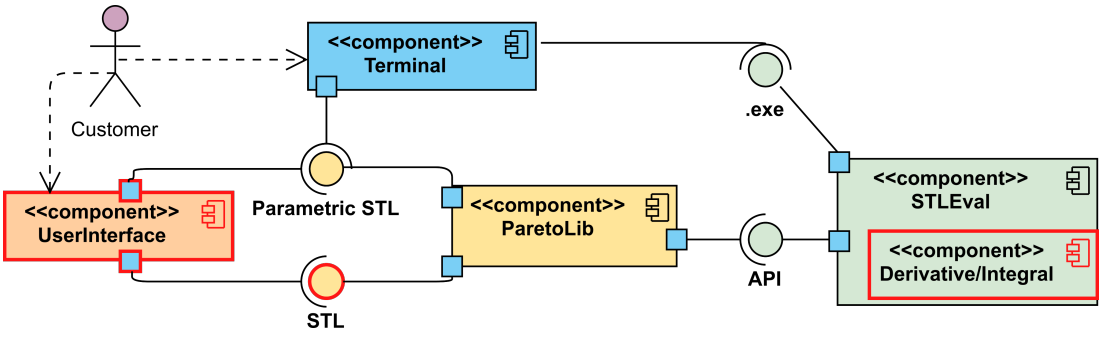

Figura 4.2: Estructura del proyecto

Como vemos en la figura 4.2, la GUI se comunica con la librería de minería ParetoLib que nos permite evaluar propiedades con parámetros, utilizando en ese caso los métodos de la librería, o sin ellos, haciendo simplemente una consulta a la API STL. A su vez ParetoLib esta conectado por binarios precomplilados con STL.

La anterior forma de hacer consultas a través de la terminal obligaba a hacerlas por separado a STLe o a la librería de minería dependiendo si quisiésemos una consulta STL paramétrica o no. La nueva GUI unifica el proceso y facilita la usabilidad de cara al usuario.

Como partes aportaciones nuevas al proyecto (marcadas en rojo) esta la propia GUI, los operadores derivada e integral en STLe y una modificación en la API OracleSTLe, con la cual conseguimos que sea factible hacer consultas no paramétricas desde ParetoLib.

4.3. Guía de uso

En esta sección vamos a describir alguna de las partes interactuables o visibles de la GUI:

- STL specification: Seleccionamos un archivo **.stl** el cual contiene una formula paramétrica o no, esto es importante porque si deseamos hacer una consulta no paramétrica debemos de ajustar la formula para que no haya ningún parámetro. Cuando seleccionemos el archivo, la descripción de la formula aparecerá en una caja inferior la cual solo sera visual y no modificable.
- Input signal: Necesita de un archivo **.csv** que contenga la especificación de la señal. Una vez seleccionado aparece un dibujo de esta en la caja grande de la ventana
- Parameters: Proporcionamos un archivo **.param** que contiene los parámetros a estudiar que aparecen en la formula del archivo .stl seleccionado. Al elegir un archivo nos aparecerá mas abajo una caja donde poder rellenar el valor mínimo y máximo de cada parámetro. Para una consulta no paramétrica no hace falta seleccionar ningún archivo.
- Parametric STL: Elegimos si queremos una consulta STL paramétrica o no, en el caso de quererla seleccionamos que si y viceversa.
- Run: Boton de ejecutar la consulta. Para poder hacerla necesitamos los 3 input files anteriores descritos (2 en el caso no paramétrico) y se nos devolverá un mapa con las regiones falsas o verdaderas según los parámetros 3.4 si la consulta es paramétrica o un True/False en el caso contrario.

4.4. Otros

Cabe mencionar, que para sacar todo el potencial de PyQt5 hemos hecho uso de la herramienta Qt Designer. Gracias a ella hemos podido construir la interfaz en sí. Una vez hecha, la unimos al archivo GUI.py para aportar funcionalidades al propio diseño.

Capítulo 5

Conclusiones

Para acabar este trabajo presentamos algunas conclusiones sobre los temas tratados, tanto teóricos como prácticos, y discutimos algunas ideas propicias para la mejora y continuidad del proyecto.

5.1. Conclusiones (español)

En vista a los resultados del capítulo anterior ...

En este proyecto, hemos actualizado los binarios y librerías dinámicas de STLEval que incorpora ParetoLib para dar soporte a las nuevas operaciones de derivación e integración. Además, hemos implementado una interfaz gráfica que abstrae la complejidad de invocar al código fuente 3.4 y resume la mayor parte de las opciones de configuración del algoritmo de minería.

5.2. Trabajo futuro

Incluir más opciones en la GUI que nos permitan configurar parámetros adicionales de ParetoLib: p.ej., (des)activar el paralelismo, precisión del aprendizaje (EPS, DELTA, STEPS) ...

Actualmente STLEval sólo soporta interpolación constante. Como trabajo futuro, se plantea extender dicha herramienta para soportar nuevos tipos de interpolación (lineal, splines, etc.). Esto abre la posibilidad de desarrollar nativamente nuevos tipos de operadores lógico-temporales que permitan realizar predicciones sobre el futuro en base a unas señales ya conocidas y recogidas del artefacto que queramos monitorizar (aproximación mediante series de Taylor, análisis estadístico de las trazas u otros métodos de aprendizaje).

Implementar un procesador de lenguaje natural que facilite la escritura de expresiones en STL en un formato más agradable. P. ej.:

- "En el futuro, la propiedad X se cumple"
- "F G (v >120)" en lugar de "(F (G (>v 120))"

Reimplementar el núcleo de ParetoLib para aumentar las prestaciones computacionales: Python se puede compilar en lugar de interpretar, lo que aumentaría el rendimiento de la librería de minería. La mejora sería mínima (ParetoLib llama a STLeval para la mayoría de los cálculos, y STLeval está en C++), pero la mejora del rendiento al compilar Python nos ayudaría ahorrar unos pocos milisegundos.

5.3. Conclusions (English)

In view of the results of the previous chapter, ...