Inferencia y verificación de propiedades en sistemas ciber-físicos estocásticos

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

Este Trabajo de Fin de Grado tiene principalmente como objetivo la ampliación de las capacidades de análisis de las herramientas para la verificación de los sistemas ciberfísicos en tiempo de ejecución. Los requisitos de la verificación están expresados en Signal Temporal Logic (STL), un tipo de lógica temporal enfocada al análisis de señales analógicas o reales. La lógica temporal es un tipo de lógica modal que nos permite expresar propiedades sobre un estado o una secuencia de estados del sistema. La primera mejora consiste en añadir dos nuevos operadores lógicos para definir tendencias (derivadas) y acumulaciones (integrales) sobre esas señales.

La segunda parte del proyecto consiste en incorporar los nuevos operadores lógicos en una una biblioteca de minería de datos ya existente para evaluar expresiones de STL paramétrica. Hemos comprobado el buen funcionamiento de los nuevos operadores y hemos diseñado una nueva interfaz gráfica para ofrecer una interacción mas fácil tanto con STL como con la biblioteca de minería.

Palabras Clave:

- Lógica Temporal
- STL
- Derivada
- Integral
- Interfaz Gráfica
- PyQt
- Minería de Datos
- C++
- Python

Abstract

The purpose of this Final Degree Project is mainly to expand the analysis capabilities of the tools for the verification of cyber-physical systems at runtime. The verification requirements are expressed in Signal Temporal Logic (STL), a type of temporal logic focused on the analysis of analog or real signals. Temporal logic is a type of modal logic that allows us to express properties about a state or a sequence of states of the system. The first improvement consists of adding two new logical operators for defining trends (derivatives) and accumulations (integrals) in those signals.

The second part of the project consists of including the new logical operators to an existing data mining library for evaluating parametric STL expressions. We have tested the functionality of the new operators and designed a new graphical interface that offers an easier interaction with both STL and the data mining library.

Keywords:

- Temporal Logic
- STL
- Derivative
- Integral
- GUI
- PyQt
- C++
- Python
- Data Mining

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 [6] (*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 [7], un tipo de lógica temporal enfocada al análisis de señales analógicas o reales.

1.2. 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** [2]: 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** [8]: Es una biblioteca 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 bibliotecas DLL de STLEval que empaqueta de forma conjunta con el resto de la biblioteca, así como la interfaz gráfica de usuario. La nueva interfaz gráfica de usuario permite interaccionar tanto con la biblioteca de aprendizaje como indirectamente con la herramienta STLEval.

1.3. 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. Finalmente, continuamos con las conclusiones más relevantes del proyecto en el capítulo 5, donde se incluye también ideas para el trabajo futuro.

1.4. 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.

1.4.1. 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.

1.4.2. 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 bibliotecas 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.

1.4.3. 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.

1.4.4. 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 biblioteca ParetoLib. Este hecho, la posibilidad de acceder tanto a la biblioteca 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 bibliotecas auxiliares para el tratamiento de datos con las que extender las funcionalidades de la interfaz en el futuro contribuyeron también a esta decisión.

1.5. 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.

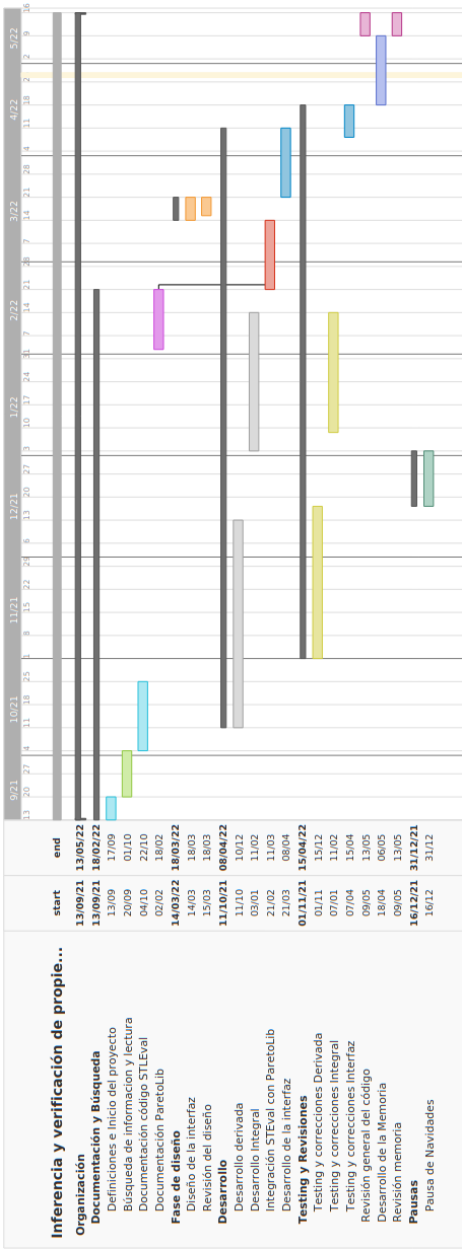

Figura 1.1: Diagrama de Gantt.

Capítulo 2

Introduction

2.1. Cyber-physical system

Cyber-physical systems are a special type of real-time systems that combine a micro-controller 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 [6] implement a monitor that monitors that the current execution meets specified requirements. Monitors act as keepers, alerting the user when execution deviates from desired behaviour 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 behaviours. 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 [7], 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* (integrals),
- 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** [2]: It is a tool capable of processing STL specifications and evaluating them on a real signal. It has received the implementation of the new temporal logic operators that calculate derivatives and integrals on a signal.
- **STLEval** [2]: 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 (GUI). The new graphical user interface allows to interact both with the learning library and indirectly with the STLEval tool.

2.3. Document Organization

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

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 accessing both the ParetoLib mining library and the STLEval tool indirectly, motivated the

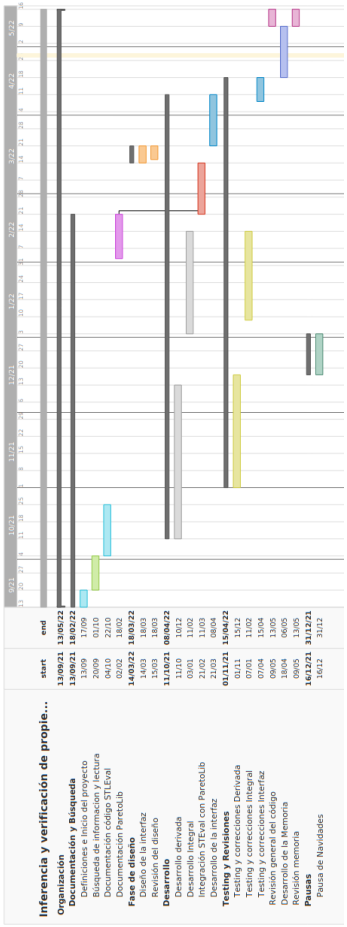

Figure 2.1: Gantt diagram.

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. 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 percepción 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 así, realizamos 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,

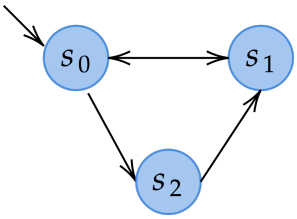

Figura 3.1: Máquina de estados.

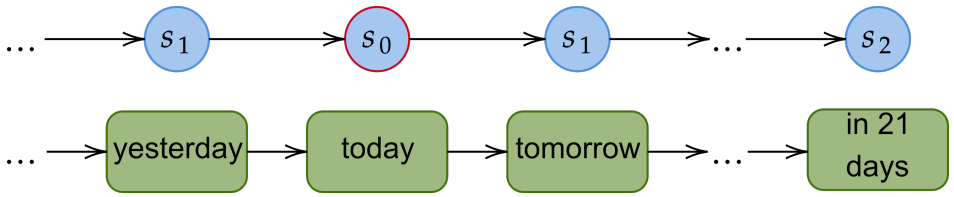

Figura 3.2: Traza de ejecución (discreta).

puede que "no todo el día esté soleado", pero quizás si por la mañana y no por la tarde o viceversa. 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 cualquier otra unidad que utilicemos para medir el tiempo, puede cambiar y tener un valor diferente. En un sistema ciberfísico, el controlador software se representa como una máquina de estados (Figura 3.1).

Tras el estado inicial s_0, la aplicación puede moverse de un estado a otro dependiendo de las restricciones de la máquina. Esto dará lugar a una traza de ejecución (Figura 3.2) donde el *tiempo* se refleja en el número de estados recorridos. En este ejemplo, el tiempo es *discreto*: cada instante temporal está asociado a un único estado del sistema.

En un entorno real, el tiempo es *denso*, es decir, el estado de la aplicación cambia en función de los datos que registra en los sensores o en las reacciones que cambian un parámetro físico (Figura 3.3).

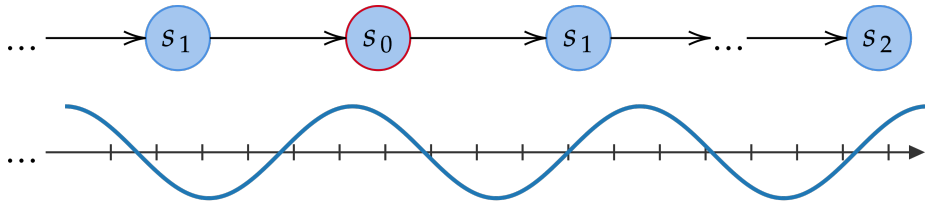

Figura 3.3: Traza de ejecución (densa).

En este trabajo utilizaremos Signal Temporal Logic [7], un tipo de lógica temporal enfocada al análisis de señales analógicas o reales.

3.2. Signal Temporal Logic

Signal Temporal Logic (STL) [7] 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 [1] o integrales en un intervalo, o calcular la derivada en un punto [5].

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

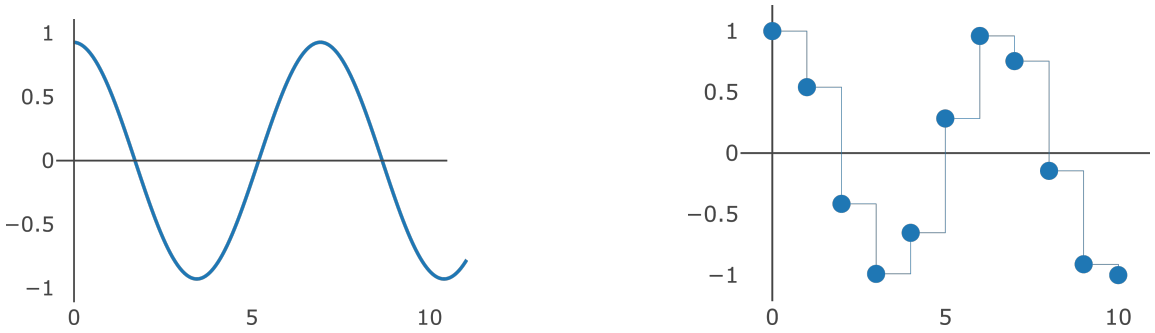

Figura 3.4: Señal original y su reconstrucción.

AMT [9] (sintaxis básica) y STLEval [2] (sintaxis 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 [5].

3.3. 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 [5].

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 3.4 ilustra la señal original y su representación interna en STLEval.

3.4. 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:

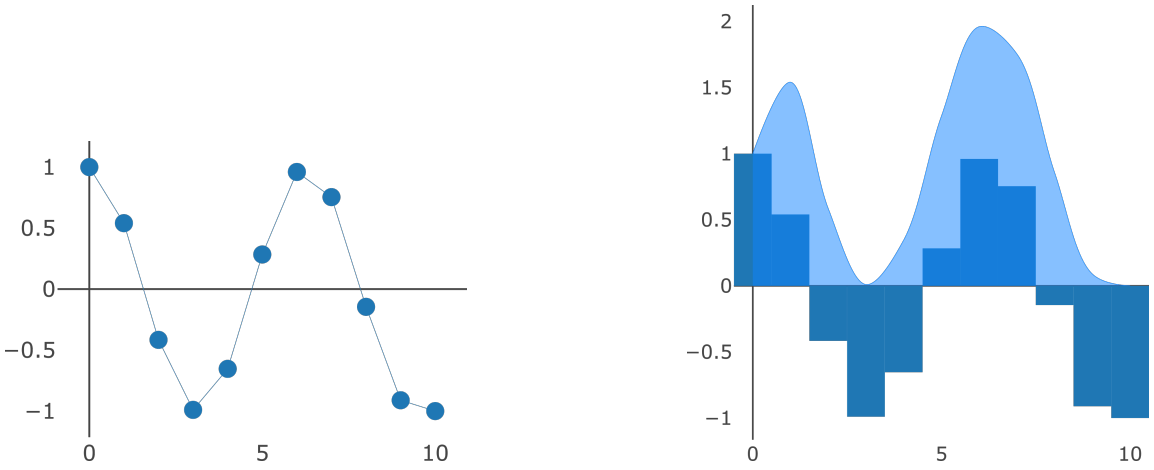

Figura 3.5: Interpretación de la derivada e integral.

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

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 3.5).

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 ($I_[a,b] \equiv \mu^i_{[a,b]}$).

3.5. Integración con ParetoLib

ParetoLib [3, 8] es una biblioteca de minería que recibe una especificación paramétrica o plantilla en STL y devuelve el rango de valores de las variables 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 tanto para Linux como para Windows, lo que convierte a la biblioteca en una herramienta multiplataforma.

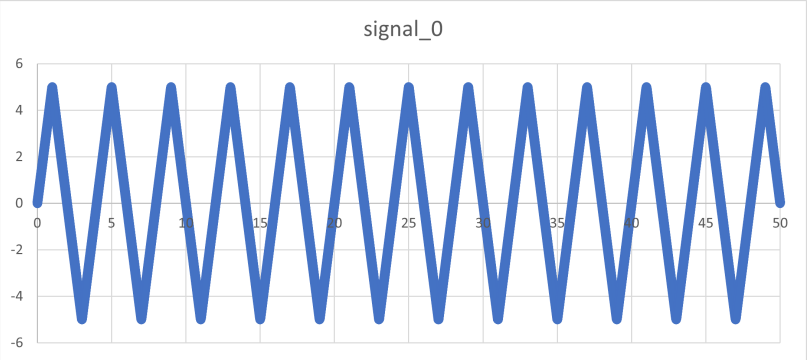

Figura 3.6: Señal triangular.

Gracias a la actualización de STLEval, podemos evaluar expresiones que involucren parámetros en derivadas e integrales. Por ejemplo, dada una señal triangular (Figura 3.6) 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 3.7 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).

Asimismo, podemos ejecutar expresiones con integrales como por ejemplo:

$$On_{[0,p_1]} I\,\mathbf{x}_0 >= 5 * 10^6 - p_2$$

donde p_1 y p_2 son los parámetros, $\mathbf{x}_0$ es la señal, I es el operador integral y $On_{[0,p_1]}$ indica el intervalo de integración. Igualmente, ParetoLib devuelve como resultado la imagen

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

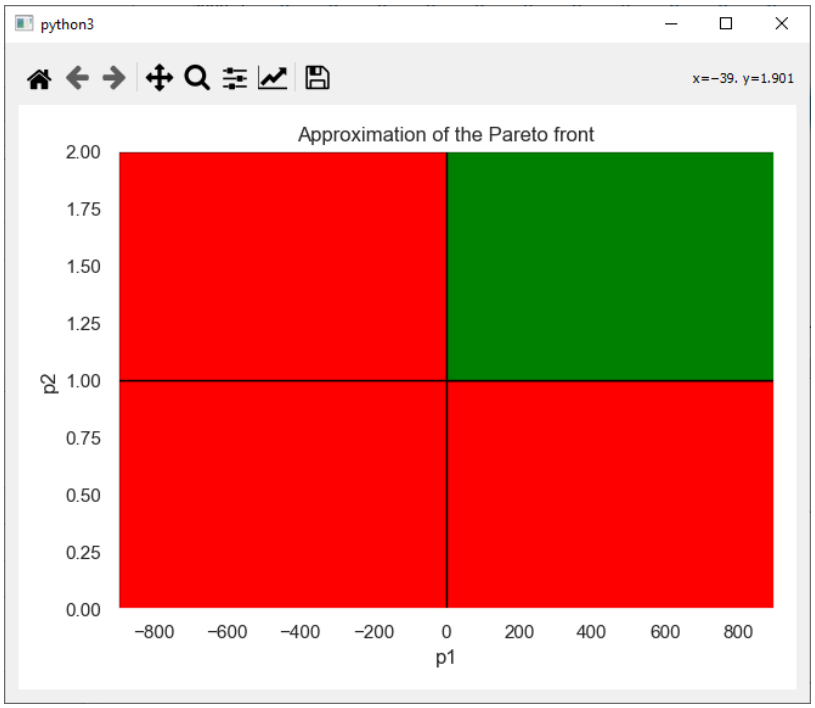

Figura 3.7: Resultado paramétrico para la derivada.

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

ParetoLib está escrito en Python. El código fuente 3.1 ilustra la forma tradicional de invocar a la biblioteca 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
```

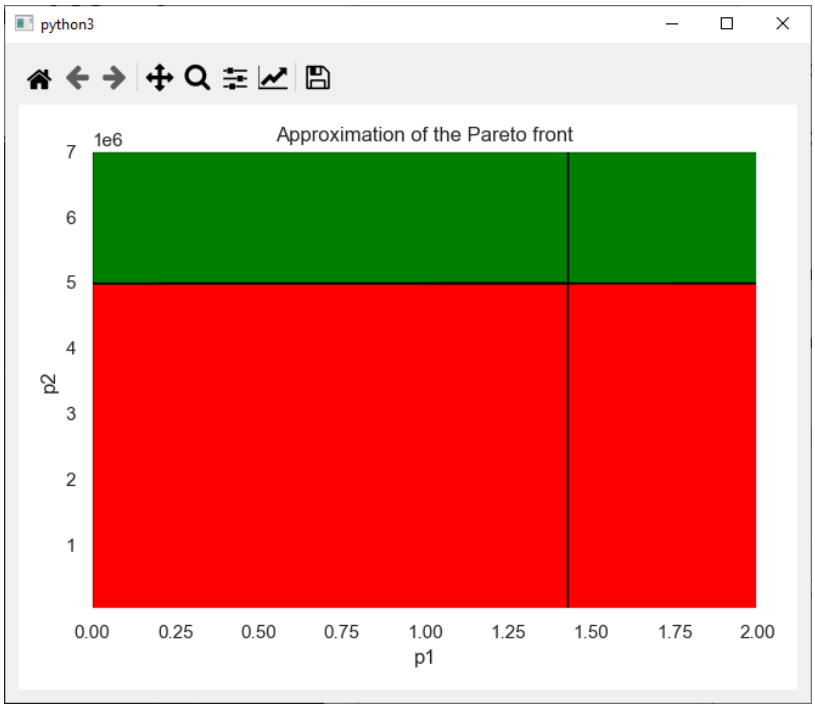

Figura 3.8: Resultado paramétrico para la integral.

```
oracle = OracleSTLeLib()
oracle.from_file(nfile, human_readable)
rs = Search2D(ora=oracle,
              min_cornerx=min_x,
              min_cornery=min_y,
              max_cornerx=max_x,
              max_cornery=max_y,
              epsilon=EPS,
              delta=DELTA,
              max_step=STEPS,
              blocking=False,
              sleep=0,
              opt_level=0,
              parallel=False,
              logging=True,
              simplify=True)
```

Código 3.1: Invocación de la biblioteca ParetoLib mediante código fuente en Python.

Capítulo 4

Interfaz gráfica

4.1. Estructura

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 biblioteca ParetoLib.

En este apartado proponemos un espacio donde el usuario tenga que simplemente adjuntar los ficheros de entrada con las señales y especificaciones, elegir la operación a realizar entre las opciones disponibles para finalmente visualizar y descargar los resultados en forma gráfica.

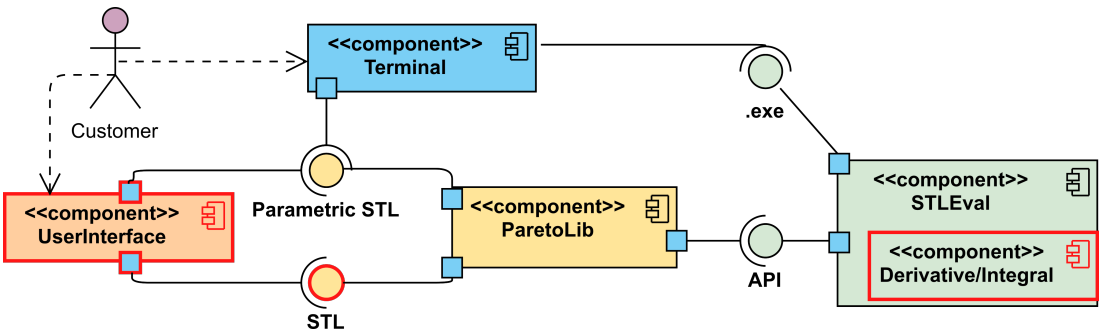

Figura 4.1: Estructura del proyecto.

Tal y como vemos en la figura 4.1, la nueva interfaz gráfica de usuario se comunica con la biblioteca de minería ParetoLib. ParetoLib nos permite evaluar propiedades con parámetros, utilizando en ese caso los métodos nativos de la biblioteca; o sin parámetros, redirigiendo

la consulta a través de una adaptación de la interfaz anterior. A su vez, ParetoLib está conectado por el otro extremo a los binarios precompilados y la API de STLEval.

Originariamente, el formato de las especificaciones en STL obligaba a interacturar con la herramienta STLEval o la biblioteca ParetoLib por separado según si las ecuaciones incluían parámetros o no (componente azul de la imagen). La nueva interfaz gráfica de usuario unifica el proceso y facilita la usabilidad de cara al usuario.

Como aportaciones nuevas al proyecto (marcadas en rojo) está la propia interfaz gráfica de usuario, los operadores derivada e integral en STLEval y una modificación en la API paramétrica de STL, con la cual conseguimos ejecutar consultas no paramétricas desde ParetoLib.

4.2. Bibliotecas utilizadas

Debido a que la biblioteca ParetoLib está escrita en Python, y que con una pequeña modificación de la interfaz de interconexión permite acceder indirectamente a la API de STLEval, hemos decidido desarrollar la interfaz gráfica de usuario en ese mismo lenguaje. La interfaz gráfica se ha testeado sobre Python 3.8 aunque ParetoLib está implementada para funcionar sobre Python 2.7 o superior. Además de las dependencias previas de ParetoLib, para la elaboración de la parte gráfica de este proyecto hemos hecho uso de las últimas versiones de estas bibliotecas adicionales:

- PyQt5
- Matplotlib
- Pandas
- Seaborn

4.2.1. PyQt5

La biblioteca PyQt5 está basada en la biblioteca gráfica Qt y sirve para realizar el diseño gráfico de la ventana principal de la aplicación desde Python. En nuestro caso optamos por un diseño simple de dos columnas: en la parte izquierda tenemos 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 tenemos la visualización de las señales tratadas y la especificación STL importada, la cual sólo se puede visualizar en modo lectura (es decir, los cambios en la especificación STL no se registrarán para la evaluación). Para extraer todo el potencial de PyQt5 hemos hecho uso de la herramienta *Qt Designer*. Gracias a ella hemos podido construir el boceto de la

interfaz, sobre la que después hemos anclado manualmente las funcionalidades a los botones correspondientes.

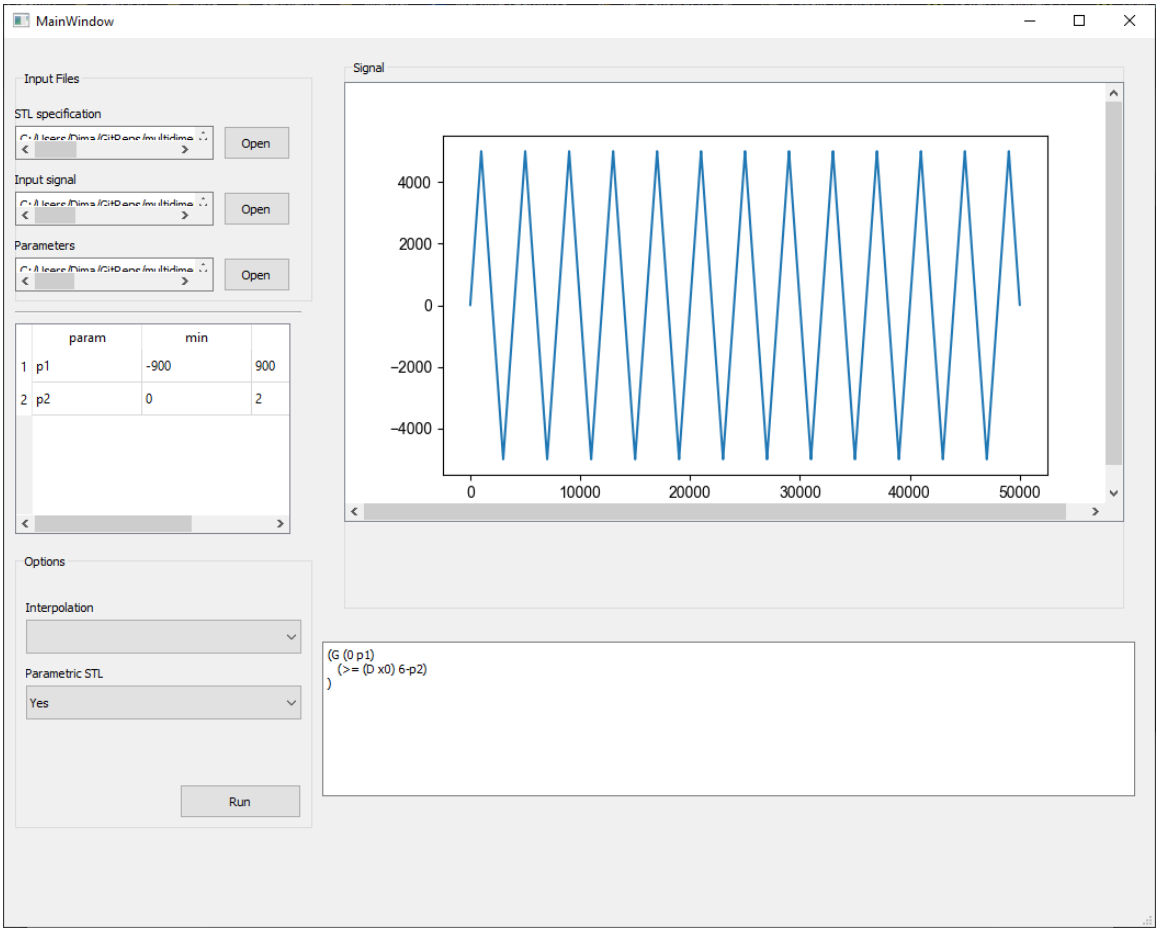

Figura 4.2: Ventana principal de ParetoLib.

4.2.2. MatplotLib

La biblioteca MatplotLib da soporte a la mayoría de las bibliotecas gráficas avanzadas en Python (p.ej., Seaborn). Desde la versión inicial de ParetoLib, accesible únicamente a nivel de código fuente en Python, MatplotLib se utiliza para generar los gráficos resultantes de la minería de propiedades STL paramétricas (Figuras 3.7–3.8). El uso del entorno PyQt5 para el diseño de la ventana principal de la nueva interfaz gráfica de usuario obliga a reajustar ligeramente la producción de estas imágenes de resultado para que se impriman dentro de una nueva ventana flotante en PyQt.

4.2.3. Pandas

La biblioteca Pandas es una biblioteca de Python especializada en el manejo y análisis de grandes bloques de datos. En nuestro caso, usamos esta biblioteca para la lectura lectura de las señales de entrada, las cuales están en ficheros csv y pueden albergar incluso miles de puntos.

Con esta biblioteca realizamos la transformación y almacenamiento de nuestras señales en una estructura de datos intermedia tipo *DataFrame*, la cual utilizamos posteriormente como dato de entrada para los métodos de visualización gráfica de la señal a través de la biblioteca Seaborn.

4.2.4. Seaborn

Basada en MatplotLib, Seaborn es una biblioteca que permite generar fácilmente elegantes gráficos. Nosotros la utilizamos para dibujar las señales a partir de los datos importados en los archivos csv.

La función de esta biblioteca es meramente estética: creemos que este recurso es principalmente un añadido atractivo a la herramienta que puede ser útil 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.

4.3. Guía de uso

En esta sección vamos a describir las partes interactuables o visibles más relevantes de la interfaz gráfica de usuario. Tras la instalación de las dependencias de ParetoLib mediante los scripts de la biblioteca, la interfaz gráfica de usuario puede arrancarse mediante la ejecución del siguiente comando desde el directorio raíz del código del repositorio:

```
user@pc:~$ python ParetoLib/GUI/GUI.py
```

Código 4.1: Inicio de la interfaz gráfica de usuario de ParetoLib.

En la ventana, veremos los botones y opciones principales:

- STL specification: Permite seleccionar un archivo **.stl**, el cual contiene una formula en STL (paramétrica o no). Cuando seleccionamos el archivo, la descripción de la formula aparece en una caja inferior en modo lectura (es decir, no modificable). Según si la consulta al motor de aprendizaje es paramétrica o no, habrá que modificar el casillero *Parametric STL*.
- Input signal: Permite seleccionar un archivo **.csv** que contiene la definición de la señal. Una vez seleccionado aparece un dibujo de esta en la caja grande de la ventana.

Figura 4.3: Resultados no paramétricos.

- Parameters: Permite seleccionar un archivo **.param** que contiene los parámetros a estudiar y que aparecen en la formula del archivo .stl seleccionado. Es un fichero opcional: para una consulta no paramétrica no hace falta seleccionar ningún archivo. Al elegir un archivo, aparecerá más abajo una caja donde se deberá rellenar el intervalo de búsqueda (valores mínimo y máximo) de cada parámetro sobre los que la biblioteca de ParetoLib realizará el aprendizaje.
- Parametric STL: Permite seleccionar si evaluaremos una consulta STL paramétrica o no: en el caso de quererla, seleccionamos que sí y viceversa.
- Run: Permite ejecutar la consulta. Para poder hacerla necesitamos los tres ficheros de entrada anteriores descritos (dos ficheros en el caso no paramétrico). El clic de este botón recoge las rutas de los ficheros de entrada, el rango de valores numéricos de los parámetros (en el caso de haberlos) y prepara la invocación a la biblioteca de minería de forma similar al código mostrado en el código fuente 3.1. En el caso de ser una consulta paramétrica, se nos devolverá un mapa con las regiones falsas o verdaderas según los parámetros (Figuras 3.7–3.8). En el caso contrario, si la consulta no es paramétrica, simplemente devolverá el resultado de la evaluación: *True* o *False* (Figura 4.3).

Capítulo 5

Conclusiones y trabajo futuro

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

Este trabajo ha tenido como objetivo mejorar las capacidades de análisis de las herramientas para la verificación de los sistemas ciberfísicos en tiempo de ejecución. Para ello, hemos utilizado la lógica temporal como formalismo para especificar las propiedades que debe satisfacer el sistema. En particular, hemos usado Signal Temporal Logic (STL), un tipo de lógica temporal enfocada al análisis de señales analógicas que provienen de sensores físicos (p.ej., la velocidad o temperatura).

Entre los resultados más relevantes en este proyecto, hemos logrado extender las capacidades de la lógica temporal STL para expresar propiedades que involucren *tendencias* (derivadas), o *acumulaciones* (integrales). Hemos implementado esos nuevos operadores lógicos en las herramientas software de verificación actuales, principalmente en la herramienta STLEval. Junto con STLEval, nos hemos ocupado también de la biblioteca ParetoLib, una biblioteca 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, por lo que ha recibido una actualización de los binarios y bibliotecas DLL de STLEval que empaqueta de forma conjunta con el resto de la biblioteca para dar soporte a las nuevas operaciones de derivación e integración.

Por último, hemos proporcionado una interfaz de usuario amigable que facilita la interacción tanto con la biblioteca de aprendizaje como indirectamente con la herramienta

STLEval. La interfaz gráfica abstrae la complejidad de invocar a STLEval a través de la consola de comandos, o mediante código fuente en Python en el caso de ParetoLib (código fuente 3.1), tal y como sucedía anteriormente hasta ahora. La interfaz gráfica resume la mayor parte de las opciones de configuración de la herramienta de verificación y del algoritmo de minería.

Finalmente, nos gustaría recalcar el arduo esfuerzo realizado en la documentación e investigación bibliográfica previa para la adquisición de los conocimientos necesarios para la elaboración de los resultados presentados en estas conclusiones.

5.2. Trabajo futuro

Como trabajo futuro, planteamos las siguientes extensiones o mejoras. Por un lado, proponemos incluir más opciones en la interfaz gráfica de usuario que nos permitan configurar parámetros adicionales de las herramientas de verificación y minería y que hasta ahora se han omitido. Por ejemplo, incorporaremos nuevos campos para seleccionar el grado de precisión del aprendizaje, el nivel de paralelismo en los cálculos en ParetoLib u otras elecciones posibles desde el código fuente en Python.

Por otro lado, actualmente STLEval sólo soporta interpolación constante para rellenar el espacio entre dos puntos de la muestra. Como trabajo futuro, se plantea enriquecer dicha herramienta para soportar nuevos tipos de interpolación (p.ej., lineal, polinomial, 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). Un prototipo de nuevo operador lógico-temporal sería el operador *probabilidad* o $\mathbf{Pr}_{\sim\lambda}\varphi$, donde $\sim \in \{<, \leq, \geq, >\}$ y $\lambda \in [0, 1]$.

En siguiente lugar, la implementación de un procesador de lenguaje natural facilitará la escritura de expresiones en STL en un formato más agradable para usuarios no experimentados. En una primera aproximación, simplemente reescribirá las ecuaciones en un formato más legible (p. ej. $F\,G\,(v > 120)$ en lugar de $(F\,(G\,(> v\,120)))$ para posteriormente permitir en futuras versiones redactar propiedades del estilo *"En el futuro, la propiedad X se cumple"*.

Por último, reimplementar el núcleo de ParetoLib aumentará las prestaciones computacionales de la biblioteca. Python ofrece mecanismos para compilar el código fuente en lugar de interpretarlo, 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 el incremento del rendimiento al compilar Python nos ayudaría a ahorrar unos pocos milisegundos.

Capítulo 6

Conclusions and future work

To finish this project, we present some conclusions on the topics covered, both theoretical and practical, and we discuss some ideas that are conducive to the improvement and continuity of the project.

6.1. Conclusions

This project has aimed to improve the analysis capabilities of the tools for the verification of cyber-physical systems at runtime. To do this, we have used temporal logic as a formalism to specify the properties that the system must satisfy. In particular, we have used Signal Temporal Logic (STL), a type of temporal logic focused on the analysis of analog signals coming from physical sensors (i.e., speed or temperature).

Among the most relevant results in this project, we have managed to extend the capabilities of STL temporal logic to express properties that involve *trends* (derivatives), or *accumulations* (integrals). We have implemented these new logical operators in the current verification software tools, mainly in the STLEval tool. Along with STLEval, we have also taken care of the ParetoLib library, a mining library that learns (in)valid configurations of the monitored system, expressed as templates or parametric specifications in STL. ParetoLib internally interfaces with STLEval to perform calculations, so it has received an update to the STLEval binaries and DLLs that it packages together with the rest of the library to support the new derivation and integration operations.

Lastly, we have provided a friendly user interface that makes it easy to interact both with the learning library and indirectly with the STLEval tool. The graphical interface abstracts the complexity of invoking STLEval via the command line, or via Python source code in the case of ParetoLib (source code 3.1), as has been the case up until now. The

graphical interface summarizes most of the configuration options of the verification tool and the mining algorithm.

Finally, we would like to emphasize the arduous effort made in the documentation and previous bibliographical research for the acquisition of the necessary knowledge for the elaboration of the results presented in these conclusions.

6.2. Future work

As future work, we propose the following extensions or improvements. On the one hand, we propose to include more options in the graphical user interface that allow us to configure additional parameters of the verification and mining tools and that have been omitted until now. For example, we will incorporate new fields to select the degree of precision of the learning, the level of parallelism in the calculations in ParetoLib or other possible choices from the source code in Python.

On the other hand, currently STLEval only supports constant interpolation to fill the space between two sample points. As future work, it is proposed to enrich this tool to support new types of interpolation (i.e. linear, polynomial, splines, etc.). This opens the possibility of natively developing new types of logical-temporal operators that allow making predictions about the future based on already known signals collected from the artefact that we want to monitor (Taylor series approximation, statistical analysis of the traces or other learning methods). A prototype of a new logical-temporal operator would be the operator *probability* or $\mathbf{Pr}_{\sim\lambda}\varphi$, where $\sim \in \{<, \leq, \geq, >\}$ and $\lambda \in [0, 1]$.

Next, implementing a natural language processor will make it easier to write expressions in STL in a more pleasing format for inexperienced users. As a first approximation, it will simply rewrite the equations in a more readable format (i.e. $F\,G\,(v > 120)$ instead of $(F\,(G\,(> v\,120)))$ to later allow in future versions write properties of the style *"In the future, property X is true"*.

Lastly, reimplementing the core of ParetoLib will increase the computational performance of the library. Python offers mechanisms to compile the source code instead of interpreting it, which would increase the performance of the mining library. The improvement would be minimal (ParetoLib calls STLeval for most calculations, and STLeval is in C++), but the performance boost when compiling Python would help us save a few milliseconds.

Bibliografía

[1] Alexey Bakhirkin y Nicolas Basset. «Specification and Efficient Monitoring Beyond STL». En: *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part II*. Ed. por Tomás Vojnar y Lijun Zhang. Vol. 11428. Lecture Notes in Computer Science. Springer, 2019, págs. 79-97. DOI: 10.1007/978-3-030-17465-1_5. URL: https://doi.org/10.1007/978-3-030-17465-1_5.

[2] Alexey Bakhirkin, Akshay Mambakam y José-Ignacio Requeno Jarabo. *StlEval*. VERIMAG Git Repository. Ver. 2.X. 2022. URL: {https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/StlEval}.

[3] Alexey Bakhirkin y col. «ParetoLib: A Python Library for Parameter Synthesis». En: *Formal Modeling and Analysis of Timed Systems - 17th International Conference, FORMATS 2019, Amsterdam, The Netherlands, August 27-29, 2019, Proceedings*. Ed. por Étienne André y Mariëlle Stoelinga. Vol. 11750. Lecture Notes in Computer Science. Springer, 2019, págs. 114-120. DOI: 10.1007/978-3-030-29662-9_7. URL: https://doi.org/10.1007/978-3-030-29662-9_7.

[4] Nicolas Basset y col. «Learning Specifications for Labelled Patterns». En: *Formal Modeling and Analysis of Timed Systems - 18th International Conference, FORMATS 2020, Vienna, Austria, September 1-3, 2020, Proceedings*. Ed. por Nathalie Bertrand y Nils Jansen. Vol. 12288. Lecture Notes in Computer Science. Springer, 2020, págs. 76-93. DOI: 10.1007/978-3-030-57628-8_5. URL: https://doi.org/10.1007/978-3-030-57628-8_5.

[5] Ali Tevfik Buyukkocak, Derya Aksaray y Yasin Yazicioglu. «Control Synthesis using Signal Temporal Logic Specifications with Integral and Derivative Predicates». En: *2021 American Control Conference, ACC 2021, New Orleans, LA, USA, May 25-28, 2021*. IEEE, 2021, págs. 4873-4878. DOI: 10.23919/ACC50511.2021.9482651. URL: https://doi.org/10.23919/ACC50511.2021.9482651.

[6] Yliès Falcone y col. «A taxonomy for classifying runtime verification tools». En: *Int. J. Softw. Tools Technol. Transf.* 23.2 (2021), págs. 255-284. DOI: 10.1007/s10009-021-00609-z. URL: https://doi.org/10.1007/s10009-021-00609-z.

[7] Oded Maler y Dejan Nickovic. «Monitoring temporal properties of continuous signals». En: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Springer, 2004, págs. 152-166.

[8] Akshay Mambakam y José-Ignacio Requeno Jarabo. *ParetoLib*. VERIMAG Git Repository. Ver. 2.X. 2022. URL: {https://gricad-gitlab.univ-grenoble-alpes.fr/verimag/tempo/multidimensional_search}.

[9] Dejan Nickovic y col. «AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic». En: *Int. J. Softw. Tools Technol. Transf.* 22.6 (2020), págs. 741-758. DOI: 10.1007/s10009-020-00582-z. URL: https://doi.org/10.1007/s10009-020-00582-z.