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 funcionales 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.

1.2. Objetivos

Los objetivos de este proyecto son:

- Extender las capacidades de la lógica temporal para expresar propiedades que involucren *tendencias* (derivadas), o *acumulaciones* (integrales), mediante el cálculo de éstas nuevas operaciones podemos realizar operaciones de predicción a futuro en base a unas señales ya conocidas y recogidas del artefacto que querramos monitorizar.
- Implementar esos nuevos operadores lógicos en las herramientas software actuales con el fin de extender su funcionamiento no sólo aplicado a todo un conjunto de señales sino también a un espacio de tiempo en particular que se encuentre dentro de nuestro conjunto principal.
- Y por último, proporcionar una interfaz de usuario amigable que facilite la interacción con dichas herramientas, para 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.

1.3. Organización

1.3.1. Documento

El documento está dividido en X capítulos. En el capítulo **??** detallaremos la semántica e implementación de los nuevos operadores lógicos. El capítulo **??** está dedicado a la nueva interfaz gráfica de usuario. Continuamos con las conclusiones más relevantes del proyecto en el capítulo **??**. Finalmente ....

1.3.2. Equipo de trabajo

Este proyecto ha sido realizado por dos estudiantes, Dymtro ap1 ap2 y Javier ap1 ap2, acompañados del tutor Jose Ignacio Requeno. Se empezó a realizar el X de septiembre del 2021 y constó de 4 fases:

Primera - Investigación

En esta fase se inició la investigación del proyecto, nos dedicamos a leer información sobre qué es la lógica temporal y la diferencia entre lógica modal, recurrimos tanto al material proporcionado por el instructor como a material externo y aprendimos un poco de su sintáxis. Esta fué la parte más complicada de todo el trabajo no sólo por la poca información que éxiste de este tema en la red, obligándonos a hacer búsquedas más exhaustivas y profundas

sobre los datos que teníamos sino también por el cambio de perspectiva que tuvimos que realizar para comprender el concepto principal de ésta lógica que es que: Una afirmación se puede convertir en negación dependiendo del tiempo en el que se evalúa la expresión.

Segunda - Documentación

Después del proceso de cambio empezamos con el de lectura y comprensión de la librería, su arquitectura y el significado de las clases además del estudio teórico de cómo funciona una integral y una derivada y cómo podemos implementarlo mediante código, en esta fase nuestro mayor dificultad fué la labor de documentación del código e interpretación del conocimiento adquirido en el apartado anterior trasladado a éste.

Tercera - Implementación

Realizamos la implementación de la integral y derivada extendiendo la librería STLe para que pueda ser aplicada a un conjunto y subconjunto de señales, para completar esta tarea realizamos una serie de pruebas con señales conocidas que nos proporcionó el tutor además de pruebas individuales realizadas con datos aleatorios y comprobando matematicamente sus resultados para comprobar que ambos, la salida de lo que había sido programado y lo que habíamos realizado a mano, sean similares.

Cuarta - Diseño y Finalización

Como última tarea creamos la interfáz gráfica de la aplicación realizando un boceto que luego intentamos trasladar a la aplicación, la tarea más complicada de esta parte fué la realización de la conexión entre los diferentes layouts de las clases de la interfaz y los métodos propios de la extensión de la libreria STLe, nos ayudó bastante el hecho de que la API ya estaba realizada por parte de la librería "multidimensional_search", al decidir codificar está parte de la aplicación con el lenguaje Python nos resulto mucho más facil las tareas de tratamiento de datos y creemos que es un lenguaje que puede facilitar la implementación de las futuras operaciones que sugeriremos al final de este documento.

1.3.3. Planificación y Organización

Debido a la dificultad matemática de este proyecto para nosotros, decidimos realizar las tareas de manera conjunta, todos los apartados de este proyecto excepto la labor de documentación se realizaron a la par por ambos integrantes del grupo.

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

Diagrama Gant **GRAFICO**

Capítulo 2

Nuevos operadores para STL

2.1. 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; 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 $t_1, \geq t_2$.

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

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

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 + 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**.

2.2. STLEval

La imagen **??** ilustra la señal original y su representación interna en STLEval.

2.2.1. Operador integral

El cálculo de la derivada e integral se aproximan mediante las siguientes expresiones matemáticas:

$\mu^i_{[a,b]} = \int_a^b g(\mathbf{x}_\tau)\delta\tau \geq c$

$\mu^d_+ = \frac{dg(\mathbf{x})}{dt^+} \geq c$ $\mu^d_- = \frac{dg(\mathbf{x})}{dt^-} \geq c$

Dado que internamente STLEval representa las señales reales como una serie numérica de tiempo discreto, aproximamos como:

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

JIComment: TODO: Rescatar las fórmulas matemáticas (sumatorios) del PDF referenciado en la cita anterior.

Gráficamente, la derivada se interpreta como la pendiente entre dos puntos de la señal; y la integral como el sumatorio de los rectángulos contenidos (Figura **??**).

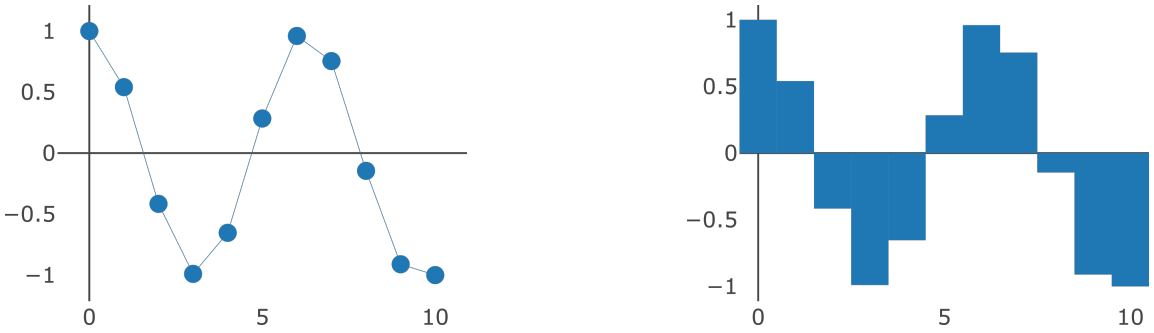

Figura 2.1: Interpretación de la derivada e integral.

2.2.2. Operador derivada

2.2.3. 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 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.

ParetoLib está escrito en Python. El código fuente **??** ilustra la forma de invocar a la librería.

Dada una señal triangular **??** y la especifiación STL X, la imagen **??** muestra en las configuraciones de los paraémtros X que satisfacen la propiedad (región en verde) y las que lo falsifican (región en rojo).

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 **??** y resume la mayor parte de las opciones de configuración del algoritmo de minería.

JIComment: TODO:

- Referenciar a la rama de ParetoLib/GUI donde se engloban los cambios de la GUI + los nuevos binarios de STLEval
- Describir un ejemplo de STL con parametros sobre un operador derivada/integral. Por ejemplo " I [0, 20] sin(x) " sería 0; "F D sin(x) >0" sería True, ...

```
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 3

Interfaz gráfica

3.1. Librerías utilizadas

Primera sección. Para el desarrollo de la parte gráfica de este proyecto hemos hecho uso de las librerías: - PyQt https://www.qt.io/qt-for-python - Matplotlib - Pandas - Seaborn

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.

Instalación: El desarrollo de la aplicación se hizo en un entorno linux haciendo uso del gestor de paquetes "pip".

1 - Antes de todo, tenemos que confirmar la versión de Python que tenemos instalada con ´python3 –version´.

2 - En el caso de que no lo tuviésemos, instalamos el gestor de paquetes "pip" con sudo 'apt-get install python3-pip' y actualizamos el mismo ´pip install -U pip´

3 - Instalamos la librería PyQT con ´pip install pyqt5´.

MatplotLib Biblioteca usada para generar gráficos a partir de datos almacenados en un array. Hacemos uso de esta librería con el fin de mostrar las señales resultantes, aplicando

Figura 3.1: Nueva pantalla de inicio.

Figura 3.2: Ejemplo.

Figura 3.3: Señal original y su reconstrucción.

las operaciones indicadas, en la sección de la aplicación propuesta para ello.

Instalación:

1 - En este caso es sumamente simple realizando: ´sudo apt-get install python3-matplotlib´.

3.2. Estructura

La interfaz gráfica, que es completamente novedosa, está montada sobre STLEval y ParetoLib y permite interactuar con ambas herramientas.

La interfaz gráfica permite evaluar tanto expresiones STL estándar (llamadas a STLEval) como minar el rango de validez de fórmulas STL con parámetros (llamadas a ParetoLib).

3.3. Configuración Técnica

El tratamiento de las señales se realiza mediante el consumo de la librería ParetoLib y el oráculo de STLe 'ParetoLib.Oracle.OracleSTLe', las funciones más importantes son:

OracleSTLeLib(stl_prop_file, csv_signal_file, stl_param_file): Constructor del objeto basado en la señal de entrada csv, sobre este objeto se van a realizar las operaciones del archivo de especificación que adjuntemos. _load_stl_formula(formula): Método que carga un archivo de especificación. eval_stl_formula(formula): Método que aplica la fórmula anteriormente cargada sobre el objeto 'oracle' actual.

3.4. Otros

Capítulo 4

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.

4.1. Conclusiones (español)

En vista a los resultados del capítulo anterior ...

4.2. 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) ...

Extender STLe para soportar nuevos tipos de interpolación (lineal, polinómica, etc.) Actualmente sólo soporta interpolación constante.

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.

4.3. Conclusions (English)

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