

Universidad Nacional de Río Cuarto
Facultad de Ciencias Exactas, Físico-Químicas y Naturales
Departamento de Computación

Trabajo Final
Modelado y Simulación de Sistemas Dinámicos con el Formalismo DEVS
Control de un Paso a Nivel de Tren

Simulación
Prof. Gonzalez, Ariel
Año 2024

Autores
Balestra, Edgar Agustín
Bernardi Quiroga, Matías

1 Introducción

Este proyecto se basa en la descripción del modelo 'Control de un Paso a Nivel de Tren' que se encuentra definido mediante el autómata temporizado:

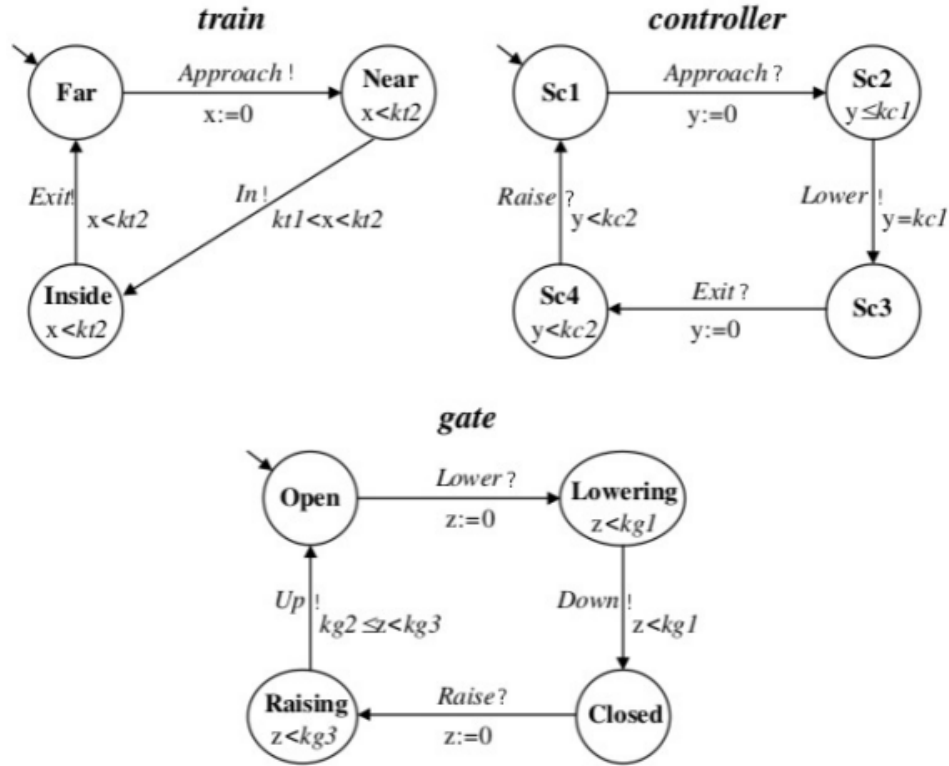


Figure 1: Autómata temporizado del sistema TCG

A partir de la definición del modelo con el autómata, se especificó el comportamiento del sistema bajo el formalismo DEVS (Discrete Event System Specification). Basándonos particularmente en el lenguaje CML-DEVS, descrito en la tesis de Hollmann. Posteriormente, se prosiguió con la implementación de los modelos DEVS en la herramienta PowerDevs, la cual nos permite simular y analizar el comportamiento del sistema de manera sencilla, precisa y eficiente.

2 Objetivos

El principal objetivo de este proyecto es analizar dos clases típicas de propiedades en el sistema: safety y liveness. Analizaremos si las propiedades que nos interesan se cumplen (o no) a través de simulaciones.

2.1 Safety

Las propiedades de safety se utilizan para especificar que "nunca va a ocurrir nada malo", garantizando que el sistema opere de manera segura en todo momento. Las mismas, pueden ser definidas contestando la siguiente pregunta: ¿qué necesito para que mi sistema no llegue a una situación no deseada? Una propiedad de safety que nos interesa analizar para el sistema TCG es: "siempre que el tren está cruzando el paso a nivel, la barrera se encuentra baja". Esta propiedad puede ser descripta utilizando la sintaxis de la Lógica Temporal Lineal: $\Box (\text{TrainInside} \rightarrow \text{GateClosed})$

2.2 Liveness

Las propiedades de liveness permiten especificar que "siempre es posible que algo bueno ocurra", asegurando que el sistema avance eventualmente hacia un estado deseado. Las mismas, pueden ser definidas contestando la siguiente pregunta: ¿qué se debe cumplir para que el sistema progrese en su ejecución? Una propiedad de liveness que nos interesa analizar en nuestro sistema TCG sería: "siempre que el tren envíe una señal Approach, en el futuro enviará una señal Exit". Es decir, el tren que en algún momento llega, eventualmente, deberá irse. Esta propiedad puede ser descripta utilizando la sintaxis de la Lógica Temporal Lineal: $\Box (\text{Approach} \rightarrow \Diamond \text{Exit})$

3 Especificación del comportamiento del sistema

Tras completar la captura de requerimientos y entender plenamente el formalismo de autómatas temporizados, se identificaron tres DEVS atómicos que definen el comportamiento del sistema TCG. Estos DEVS atómicos fueron especificados detalladamente utilizando el lenguaje CML-DEVS, tal como se describe en la tesis "Traducción y Validación Sistemática y Automática de Modelos DEVS Abstractos" de Diego Ariel Hollmann.

3.1 Especificacion en CML-DEVS

3.1.1 Train

```

1 atomic TRAIN is <X, Y, S,  $\delta_{\text{int}}$ ,  $\delta_{\text{ext}}$ ,  $\lambda$ , ta> where
2   X is
3     in :  $\mathbb{R}$ ;
4   end X
5   Y is
6     out : {exit, approach, in};
7   end Y
8   S is
9     s : {far, near, inside}  $\times$  Time;
10  end S
11   $\delta_{\text{int}}((\text{st}, \sigma))$  is
12    defcases
13      case s = (near, U(kt1, kt2)) if st = far;
14      case s = (inside, U(0, kt2)) if st = near;
15      case s = (far,  $\infty$ )         if st = inside;
16    end defcases
17  end  $\delta_{\text{int}}$ 
18   $\delta_{\text{ext}}((\text{st}, \sigma), e, (x, p))$  is
19    defcases
20      case s = (st, 0) if st = far;
21      otherwise s = (st,  $\sigma - e$ )
22    end defcases
23  end  $\delta_{\text{ext}}$ 
24   $\lambda((\text{st}, \sigma))$  is
25    defcases
26      case out = approach if st = far;
27      case out = in       if st = near;
28      case out = exit     if st = inside;
29    end defcases
30  end  $\lambda$ 
31  ta((st,  $\sigma$ )) is
32     $\sigma$ ;
33  end ta
34 end atomic

```

3.1.2 Controller

```

1  atomic CONTROLLER is <X, Y, S,  $\delta_{\text{int}}$ ,  $\delta_{\text{ext}}$ ,  $\lambda$ , ta> where
2    X is
3      in : {approach, exit};
4    end X
5    Y is
6      out : {raise, lower};
7    end Y
8    S is
9      s : {Sc1, Sc2, Sc3, Sc4}  $\times$  Time;
10   end S
11    $\delta_{\text{int}}((\text{st}, \sigma))$  is
12     defcases
13       case s = (sc3,  $\infty$ )      if st = sc2;
14       case s = (sc1,  $\infty$ )      if st = sc4;
15     end defcases
16   end  $\delta_{\text{int}}$ 
17    $\delta_{\text{ext}}((\text{st}, \sigma), e, (x, p))$  is
18     defcases
19       case s = (Sc2, U(0, kc1)) if st = Sc1 and x = approach;
20       case s = (Sc4, U(0, kc2)) if st = Sc3 and x = exit;
21     end defcases
22   end  $\delta_{\text{ext}}$ 
23    $\lambda((\text{st}, \sigma))$  is
24     defcases
25       case out = lower      if st = Sc2;
26       case out = raise      if st = Sc4;
27     end defcases
28   end  $\lambda$ 
29   ta((st,  $\sigma$ )) is
30      $\sigma$ ;
31   end ta
32 end atomic

```

3.1.3 Gate

```

1  atomic GATE is <X, Y, S,  $\delta_{\text{int}}$ ,  $\delta_{\text{ext}}$ ,  $\lambda$ , ta> where
2    X is
3      in : {raise, lower};
4    end X
5    Y is
6      out : {up, down};
7    end Y
8    S is
9      s : {open, closed, lowering, raising}  $\times$  Time;
10   end S
11    $\delta_{\text{int}}((\text{st}, \sigma))$  is
12     defcases
13       case s = (closed,  $\infty$ )    if st = lowering;
14       case s = (open,  $\infty$ )      if st = raising;
15     end defcases
16   end  $\delta_{\text{int}}$ 
17    $\delta_{\text{ext}}((\text{st}, \sigma), e, (x, p))$  is
18     defcases
19       case s = (lowering, U(0, kg1))  if st = open and x = lower;
20       case s = (raising, U(kg2, kg3)) if st = closed and x = raise;
21     end defcases
22   end  $\delta_{\text{ext}}$ 
23    $\lambda((\text{st}, \sigma))$  is
24     defcases
25       case out = down  if st = lowering;
26       case out = up    if st = raising;
27     end defcases
28   end  $\lambda$ 
29   ta((st,  $\sigma$ )) is
30      $\sigma$ ;
31   end ta
32 end atomic

```

4 Implementación del Modelo TCG

El modelo TCG fue implementado conforme a la especificación del comportamiento del sistema previamente descrita, utilizando la herramienta PowerDEVS de la siguiente manera:

1. Definición de Modelos DEVS Atómicos:

- **Tren:** Se creó un modelo DEVS atómico para representar el comportamiento de los trenes.
- **Controlador:** Se desarrolló un modelo DEVS atómico para gestionar las señales de la barrera y el control del tráfico ferroviario.
- **Barrera:** Se implementó un modelo DEVS atómico para simular el funcionamiento de las barreras de seguridad.

2. Arribos de los Trenes:

- Los arribos de los trenes se modelaron utilizando una función de distribución normal proporcionada por PowerDEVS, permitiendo una representación realista de los tiempos de llegada.

3. Tiempos de Estado del Sistema:

- Los tiempos en los que el sistema permanece en cada estado se representaron mediante una función de distribución continua. Esta función fue implementada en el lenguaje C++ por los autores de este informe y posteriormente importada como una biblioteca en los diferentes modelos atómicos para su utilización.

Esta implementación garantiza que el comportamiento del sistema TCG sea simulado de manera precisa y eficiente, permitiendo un análisis detallado de su funcionamiento.

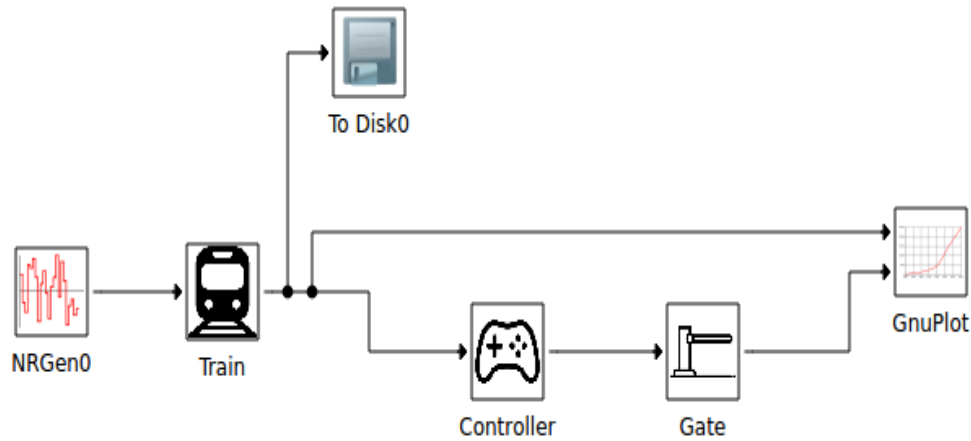


Figure 2: TCG en herramienta PowerDevs

5 Simulaciones

A continuación, se muestran los distintos análisis alcanzados bajo las simulaciones realizadas con la herramienta PowerDEVS. Se describirá como los diferentes valores que tomen los parámetros del modelo afectarán al funcionamiento del mismo. Se utilizó el atomic de GNUPlot provisto para representar de manera clara y sencilla el comportamiento del sistema a través de la ejecución de las simulaciones.

5.1 Tiempos de Señalización y Respuesta

En las siguientes simulaciones, se modificaron los valores que toman las diferentes constantes que se encuentran en el proyecto.

5.1.1 Primer experimento

En esta simulación, incrementamos el valor de la constante $kc1$, donde el tiempo que el sistema puede estar en el estado $Sc2$ del controlador es mayor. Analizaremos de manera gráfica que ocurre si el controlador desde que recibe la señal Approach del tren, tarda un poco más en mandar la señal de Lower a la barrera.

Los valores que toman las constantes del sistema en esta simulación son los siguientes:

$kt1 = 2$, $kt2 = 5$, $kc1 = 3$, $kc2 = 1$, $kg1 = 1$, $kg2 = 1$ y $kg3 = 2$

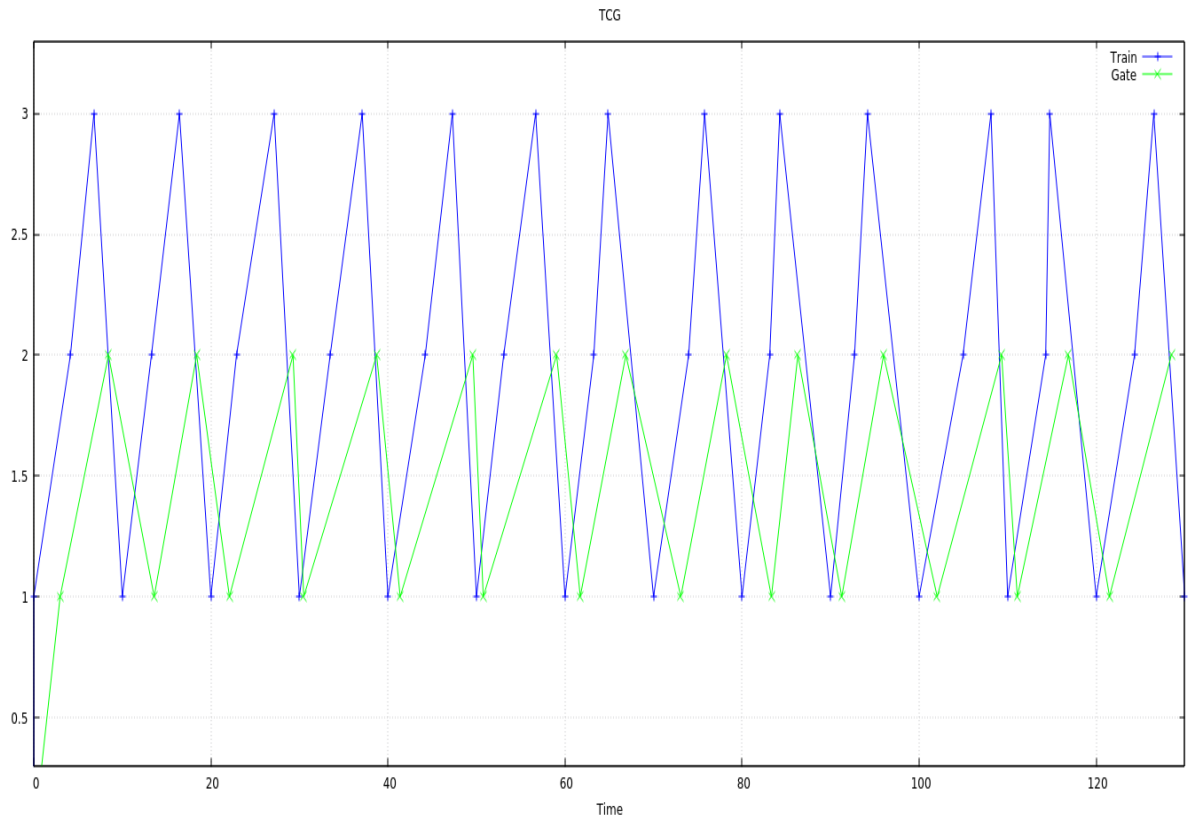


Figure 3: Simulación retrasando al controlador a mandar la señal de bajar la barrera

Como se logra observar en el gráfico generado por el atomic de GNUPlot, a partir del segundo tren que llega, la barrera se baja posteriormente a la llegada del mismo, lo cual es un error en el sistema y podría provocar una tragedia si hablásemos de un sistema real. Por lo cual, es crucial que esta constante tome valores mas bajos que 3 para que el sistema se comporte de manera adecuada. Con los parámetros previamente mencionados, la propiedad de Safety "siempre que el tren está cruzando el paso a nivel, la barrera se encuentra baja", se rompe.

5.1.2 Segundo experimento

En esta simulación, incrementamos el valor de la constante $kg1$, donde la barrera tiene un retraso mayor en el tiempo que puede estar en el estado lowering, es decir, tarda un poco mas en bajarse por completo.

Los valores que toman las constantes del sistema en esta simulación son los siguientes:

$kt1 = 2$, $kt2 = 1$, $kc1 = 1$, $kc2 = 1$, $kg1 = 4$, $kg2 = 1$ y $kg3 = 2$

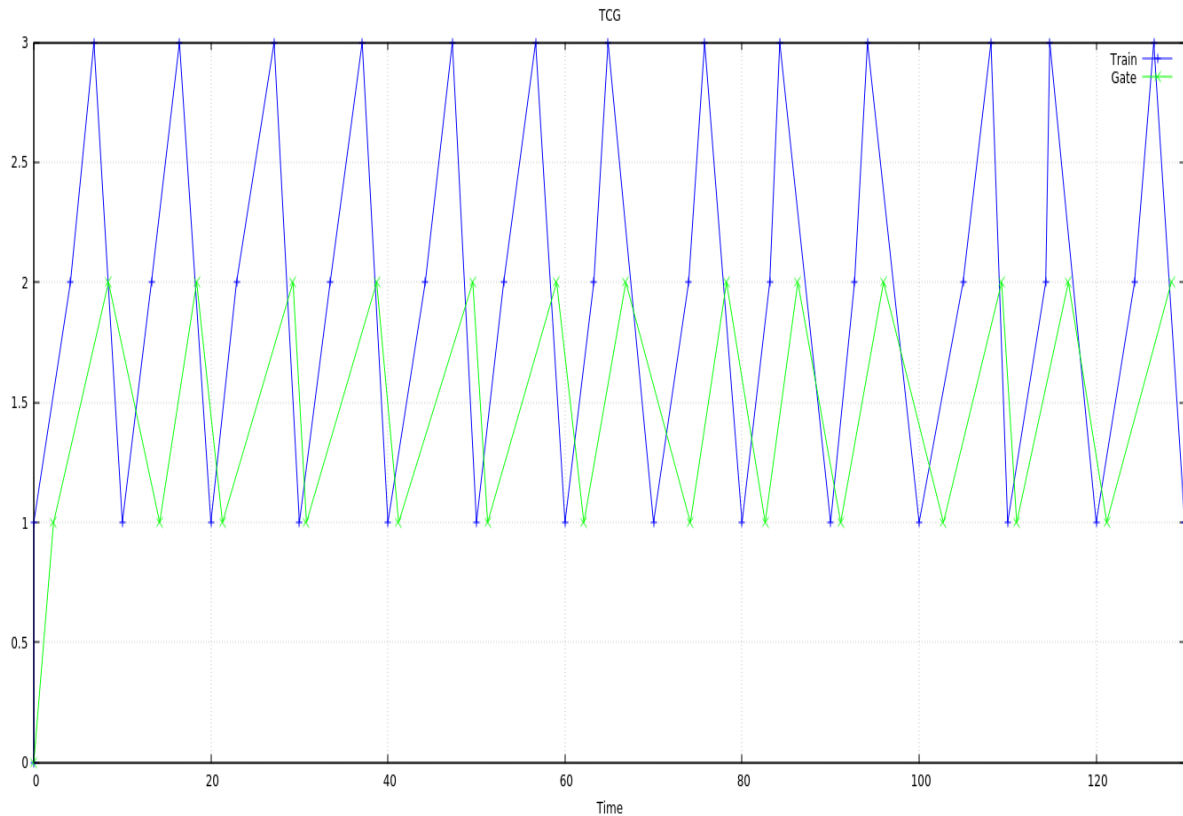


Figure 4: Simulación retrasando el tiempo que tarda la barrera en bajar

Como se logra observar en el gráfico, al igual que el experimento anterior, el incremento en el tiempo que tarda la barrera en bajarse por completo, podría desencadenar en un accidente ya que la barrera se encuentra subida cuando el tren esta cruzando.

Como se logra apreciar en el gráfico, a la segunda llegada de un tren, cuando este ingresa, la barrera aun no estaba bajada completamente. Tomando estos parámetros, la propiedad de Safety "siempre que el tren está cruzando el paso a nivel, la barrera se encuentra baja", se viola.

5.2 Frecuencia de Llegada de Trenes

En estas simulaciones, se alteró el periodo de la función de la distribución normal, la cual simula la frecuencia de los interarribos de los trenes.

5.2.1 Primer experimento

En esta primera simulación, setearemos el valor T (periodo) del DEVS atomic de la función de distribución normal en $T=1$, esto representa una llegada de trenes con una frecuencia demasiado rápida.

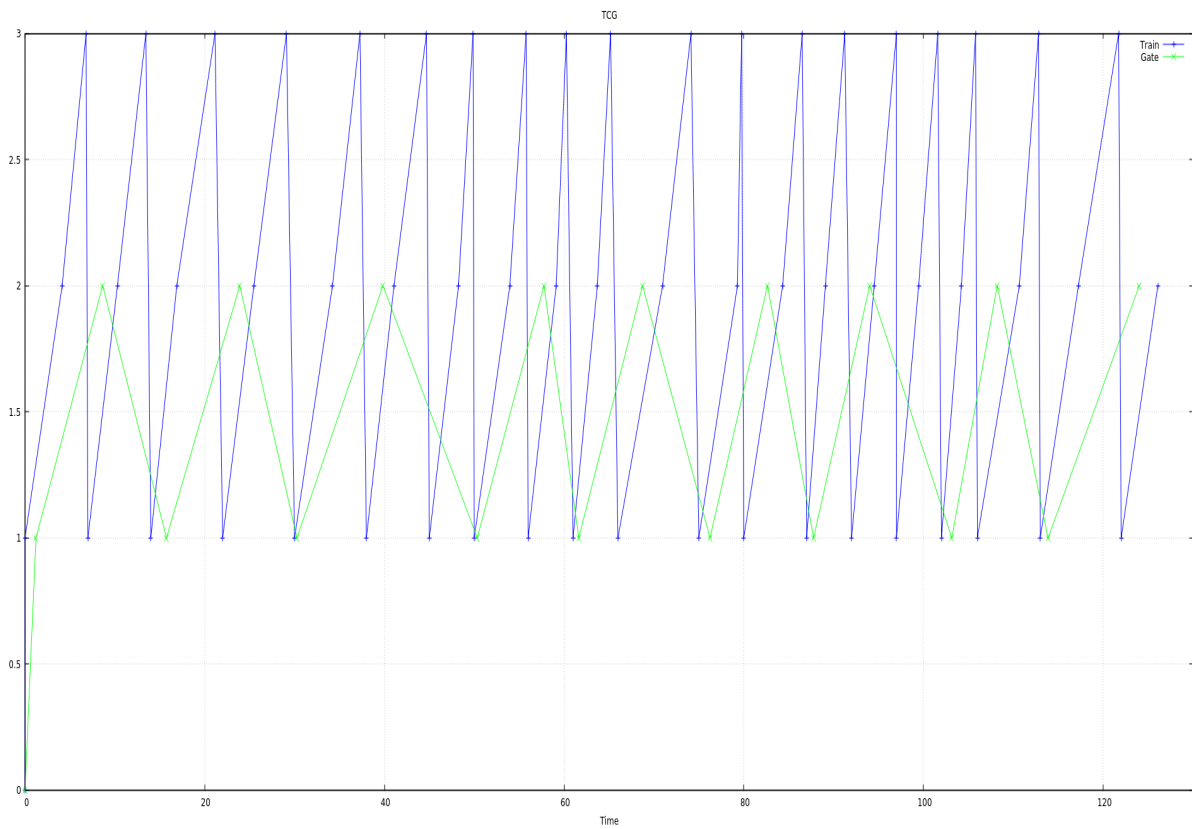


Figure 5: Simulación con interarribos de trenes cercanos

Una vez realizada la simulación, podemos observar que los trenes llegan demasiado rápido y no dan tiempo a la barrera (ni al controlador) a hacer el ciclo de subir cuando un tren se va y bajar cuando llegue el próximo.

Lo que se aprecia en el gráfico, es que un tren se aproxima, se baja la barrera, el tren ingresa, luego se va, llega otro, se va y recién en ese momento la barrera se sube.

A diferencia de los experimentos anteriores, no se logra notar con simulación, la violación

de propiedades de Safety. En cambio, se nota claramente como se viola una propiedad de liveness que nos interesa analizar en el sistema: "Cada vez que un tren se va, la barrera debe subirse hasta la espera de un nuevo arribo de otro tren".

5.2.2 Segundo experimento

Intentando que el sistema funcione de manera correcta, incrementaremos el valor del periodo de la función normal, donde ahora los trenes no llegarán uno tan seguido del otro. El valor que tomara T en esta simulación es 5.

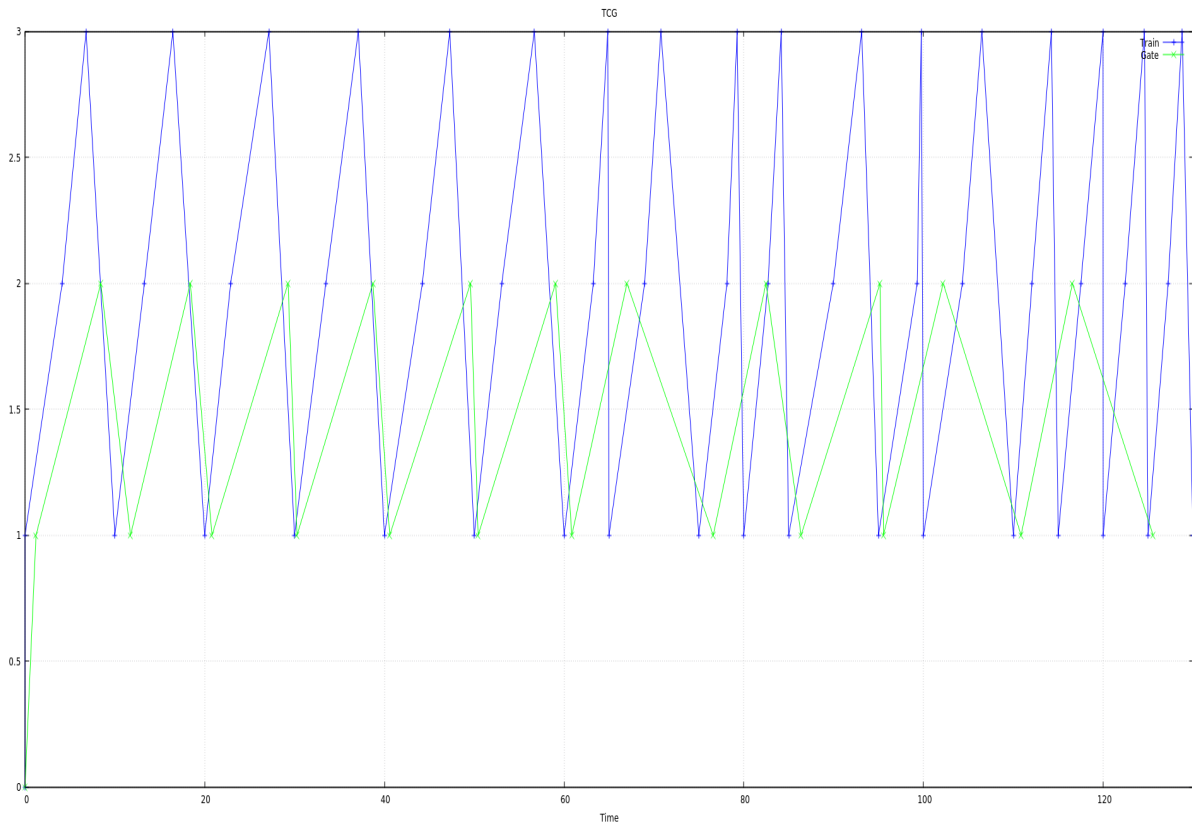


Figure 6: Simulación con interarribos de trenes mas espaciados

Nuevamente, podemos notar que el valor escogido para el periodo de la función de distribución normal no es suficiente.

Se aprecia que la simulación comienza bien, comportándose correctamente, pero pasadas las 60 unidades de tiempo, se logra visualizar una violación de la propiedad de Safety: "siempre que el tren está cruzando el paso a nivel, la barrera se encuentra baja". Esto se debe a que, como se aprecia en el gráfico, la barrera recién se termina de levantar

cuando ya un nuevo tren se esta aproximando y no le da tiempo a bajar antes de que el mismo ingrese, sino que en cambio, el tren entra, se va, y cuando otro nuevo tren se esta aproximando, recién ahí se baja.

5.2.3 Tercer experimento

Tratando de encontrar el valor adecuado para que el sistema funcione de manera correcta, fuimos incrementando de a 1 el valor del periodo hasta que el gráfico no presente inconsistencias. El mínimo valor que debe tomar la función de distribución normal que representa los interarribos de los trenes al sistema es $T = 10$.

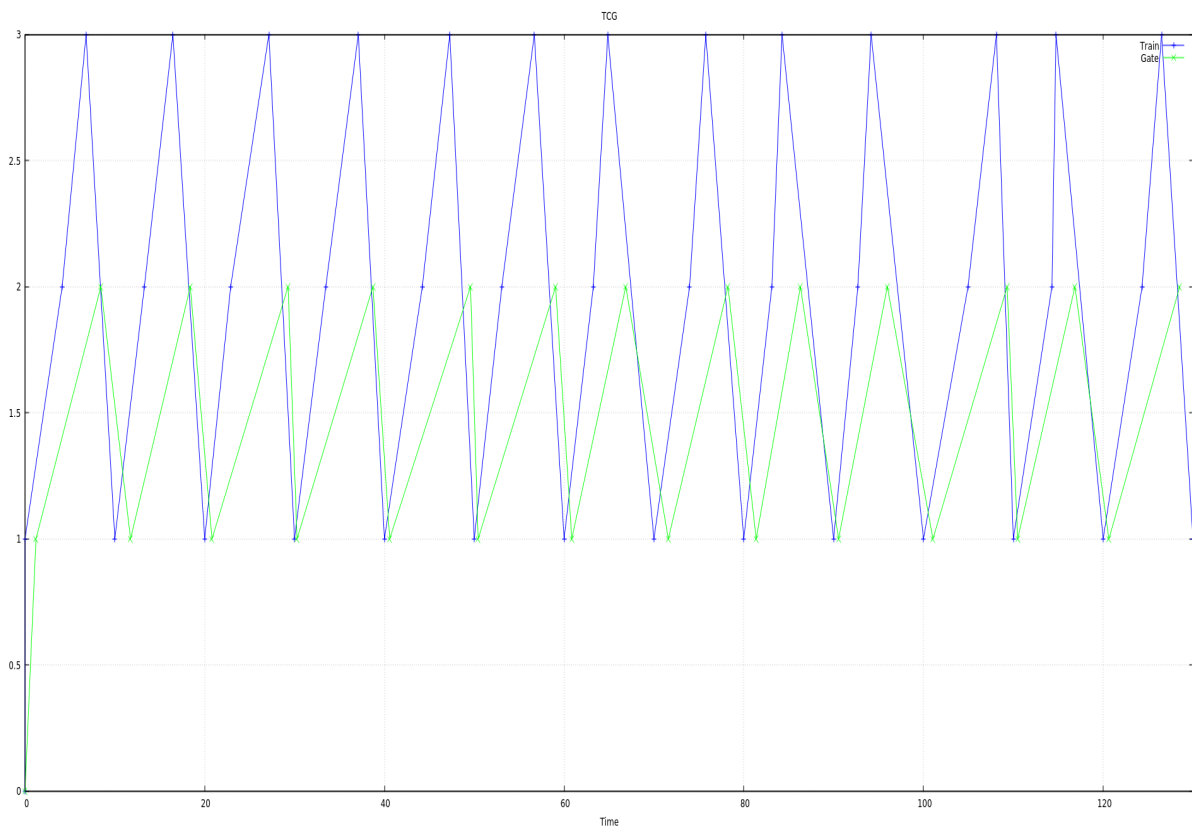


Figure 7: Simulación con los interarribos y parámetros correctos

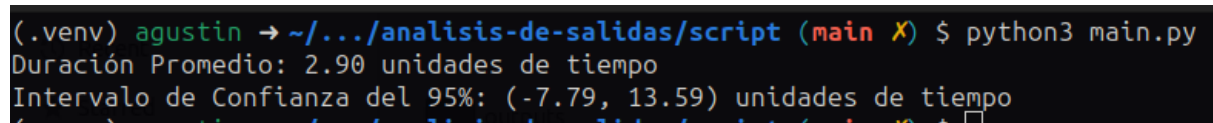
Ahora sí, en el gráfico se puede ver claramente como no se viola ninguna propiedad de Safety ni de Liveness, mostrando que el sistema funciona de manera correcta y que siempre llega a un estado deseado. Siempre que un tren cruza, la barrera se encuentra baja, siempre que un tren entra al sistema, eventualmente se va y siempre que un tren se va, la barrera se baja hasta la aproximación de un nuevo tren.

6 Análisis de salida

Para determinar el tiempo promedio que un tren permanece en el sistema, desde que envía una señal de "Approach" hasta que envía una señal de "Exit", se llevó a cabo un análisis de salidas utilizando el método del intervalo de confianza. Este análisis se realizó a partir de varias simulaciones en las que se registró el tiempo de entrada y salida de los trenes.

Se realizaron 10 simulaciones y se guardaron los datos de salida en archivos CSV. Posteriormente, se desarrolló un script en Python para procesar estos datos, calcular el tiempo promedio que un tren permanece en el sistema y determinar el intervalo de confianza del 95% para dicha duración.

El script combina los datos de todas las simulaciones, extrae los tiempos correspondientes a las señales de "Approach" y "Exit", calcula la duración de la permanencia en el sistema y finalmente, estima el intervalo de confianza. La figura 8 muestra el resultado de este análisis.



```
(.venv) agustin -> ~/.../analisis-de-salidas/script (main X) $ python3 main.py
Duración Promedio: 2.90 unidades de tiempo
Intervalo de Confianza del 95%: (-7.79, 13.59) unidades de tiempo
```

Figure 8: Output del script

A continuación, se detalla el código implementado para el mencionado análisis:

```
1 import pandas as pd
2 import numpy as np
3 from scipy import stats
4
5 # Leer archivos CSV
6 files = [f'outputs/output-train{i}.csv' for i in range(10)]
7 dataframes = [pd.read_csv(file, header=None) for file in files]
8
9 # Combinar dataframes
10 data = pd.concat(dataframes, ignore_index=True)
11 data.columns = ['Time', 'Event']
12
13 # Extraer tiempos para cada evento
14 approach_times = data[data['Event'] == 1]['Time'].values
15 exit_times = data[data['Event'] == 3]['Time'].values
16
17 # Calcular duraciones de Approach a Exit
18 durations = exit_times - approach_times[:len(exit_times)]
19
20 # Calcular duracion promedio
21 average_duration = np.mean(durations)
22
23 # Calcular intervalo de confianza
24 confidence_level = 0.95
25 degrees_freedom = len(durations) - 1
26 sample_mean = np.mean(durations)
27 sample_standard_error = stats.sem(durations)
28 confidence_interval =
29     stats.t.interval(    confidence_level,
30                         degrees_freedom,
31                         sample_mean,
32                         sample_standard_error)
33
34 # Imprimir resultados
35 print(f"Duracion Promedio: {average_duration:.2f}
36       unidades de tiempo")
37
38 print(f"Intervalo de Confianza del 95%:
39       ({confidence_interval[0]:.2f},
40       {confidence_interval[1]:.2f}) unidades de tiempo")
```

7 Conclusiones finales

La realización de este proyecto ha permitido destacar la capacidad de la herramienta PowerDEVS como intermediaria esencial entre el programador y el cliente. Mediante la misma, se pueden simular y ajustar los diferentes parámetros de un modelo complejo como lo es el de Control de un Paso a Nivel de Tren. Esto, aplicado al mundo real, es de suma importancia, donde la precisión y la seguridad en la implementación de sistemas de control de trenes pueden literalmente salvar vidas.

El uso adecuado de herramientas de simulación permite anticipar, analizar y ajustar parámetros críticos sin la necesidad de altos costos económicos ni poner en riesgo vidas humanas. En lugar de realizar pruebas directas con trenes reales y exponiendo a las personas a potenciales peligros, las simulaciones ofrecen un medio seguro y efectivo para validar y verificar estos sistemas.

Así mismo, un gran poder conlleva una gran responsabilidad. Es necesario que los programadores utilicen estas herramientas de manera adecuada, ya que tienen en sus manos la responsabilidad de garantizar la seguridad de mucha gente. A lo largo de la historia, ha habido numerosos accidentes catastróficos relacionados con fallos en el software de control. Por mencionar algunos ejemplos, el desastre de Therac-25, un acelerador lineal de radioterapia, causó la muerte de varios pacientes debido a errores en su software de control. Otro trágico incidente fue el vuelo 501 de Ariane 5, donde un error en el software de navegación resultó en la explosión del cohete poco después del despegue.

La simulación no solo ofrece una forma de minimizar estos riesgos, sino que también permite a la sociedad beneficiarse de sistemas más seguros y eficientes. Con un estudio y uso adecuado, las herramientas de simulación pueden reducir significativamente los riesgos asociados con fallos en sistemas críticos, ofreciendo una mayor protección y confiabilidad en los mismos.