Programación Concurrente 2022
Trabajo Práctico Final

Condiciones

- El trabajo es grupal, de 5 alumnos (4 es la excepción).
- La defensa del trabajo se coordinará con el grupo respecto a modalidad presencial/ videoconferencia.
- La evaluación es individual (hay una calificación particular para cada integrante)
- Solo se corrigen los trabajos que hayan sido subidos al aula virtual (LEV)
- Los problemas de concurrencia deben estar correctamente resueltos y explicados.
- El trabajo debe implementarse en lenguaje Java.
- Se evaluará la utilización de objetos y colecciones, como así también la explicación de los conceptos relacionados a la programación concurrente.

Red de Petri

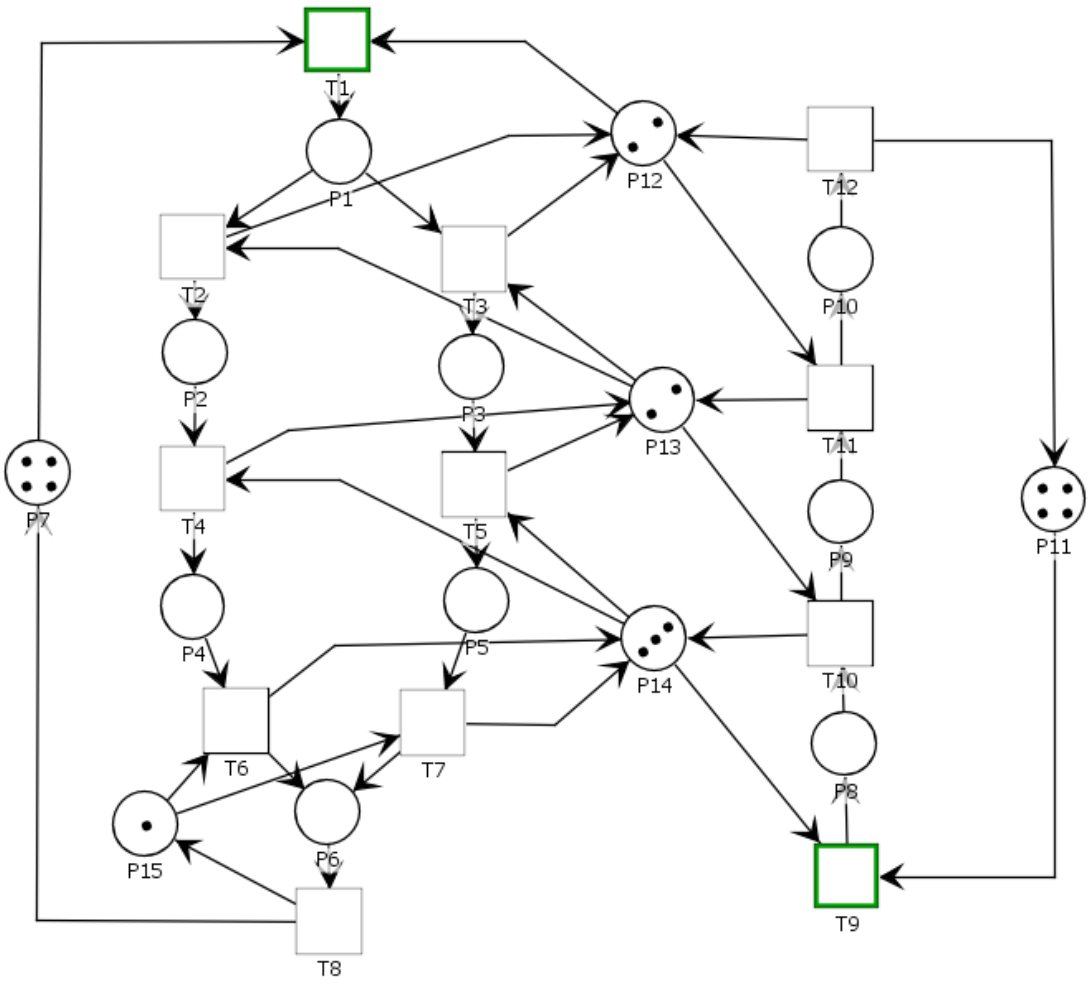

Figura 1

Enunciado

En la Figura 1 se observa una red de Petri con el modelado de un sistema de producción (la semántica del sistema debe asignarla el grupo por ej: industrial, informático, cyber physical systems, etc). Las plazas {P12, P13, P14, P15} representan recursos compartidos en el sistema. Las plazas {P7, P11} son plazas idle, que corresponden a buffers del sistema. Las plazas {P1, P2, P3, P4, P5, P6, P8, P9, P10} son plazas donde se realizan actividades relacionadas con el proceso. Es necesario que el grupo defina la semántica del sistema, es decir, si las plazas P7 y P11 representan buffers, y las plazas P12, P13, P14 y P15 recursos, por ej. Maquinas o robots, es necesario definir qué actividades se realizan en las plazas P1, P2, P3, P4, P5, P6, P8, P9 y P10.

Propiedades de la Red

- Es necesario determinar con una herramienta (simulador Ej: Petrinator, Pipe), las propiedades de la red (libre de deadlock, vivacidad, seguridad). Cada propiedad debe ser interpretada para esta red particular.
- Indicar cual o cuales son los invariantes de plaza y los invariantes de transición de la red. Y realizar una breve descripción de lo que representan del modelo de acuerdo a la semántica decidida por el grupo.
- En caso que la red tenga Interbloqueo, (deadlock):
    - Deberá agregar las plazas de control que considere hasta desbloquearla, cumpliendo con las siguientes prioridades.
        - La red debe mantener sin cambio alguna sus Invariantes de Transición (es decir misma cantidad y mismos invariantes).
        - La red debe tener el mayor grado de paralelismo posible, entendiendo por paralelismo la realización de acciones simultáneas (plazas marcadas que no sean recursos o idle).

      TIP: Para esto es posible con la herramienta Petrinator, buscar todos los posibles marcados de la red, y luego analizar solo las plazas donde se realizan actividades del proceso. Comparar los posibles marcados (numero de token promedios, cantidad de marcados, etc) de la red original vs la red desbloqueada.

Implementación

Es necesario implementar un monitor de concurrencia para la simulación y ejecución del modelo:

- Realizar una tabla, con los estados del sistema.
- Realizar una tabla, con los eventos del sistema.
- Determinar la cantidad de hilos necesarios para la ejecución del sistema con el mayor paralelismo posible:

- Caso 1: si el invariante de transición no tiene conflictos, un hilo debe estar encargado de la ejecución de las transiciones de ese invariante.
- Caso 2: si el invariante de transición tiene un conflicto, con otro invariante, debe haber un hilo encargado de la ejecución de la/s transición/es anterior/es al conflicto y luego un hilo por invariante.
- Caso 3: si el invariante de transición presenta un join, con otro invariante de transición, luego del join debe haber tantos hilos, como token simultáneos en la plaza, encargados de las transiciones restantes dado que hay un solo camino.
- Realizar un gráfico donde pueda observarse las responsabilidades de cada hilo con diferentes colores. A modo de ejemplo se observa en la figura 2 una red y cómo presentar lo solicitado, las flechas coloreadas representan cada hilo.

TIP: consultar artículo referido a determinar cantidad de hilos titulado: “Algoritmos para determinar cantidad y responsabilidad de hilos en sistemas embebidos modelados con Redes de Petri S^3PR” disponible en https://www.researchgate.net/publication/358104149

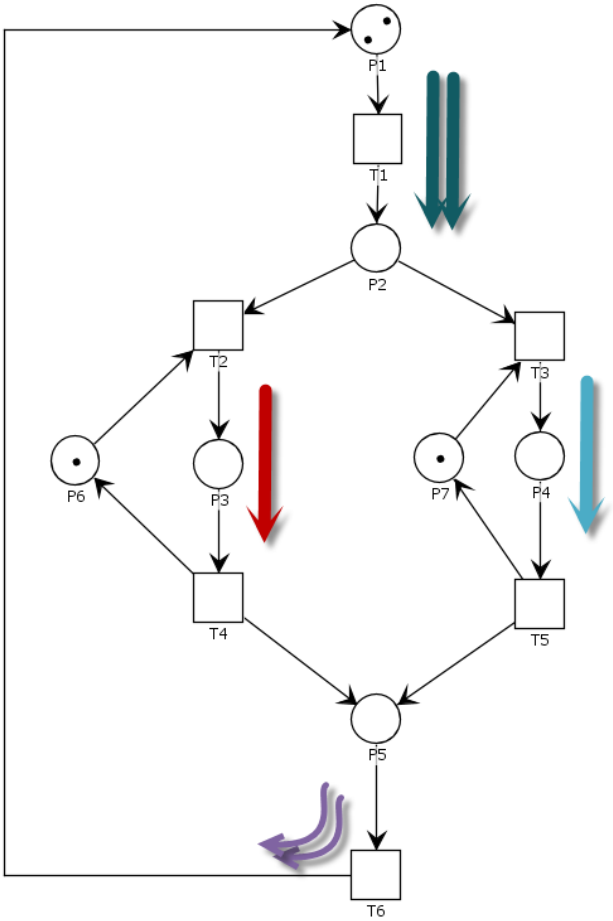

Figura 2

Tiempo

Una vez implementado el monitor de concurrencia, con el sistema funcionando, se deberá implementar la semántica temporal. Las transiciones {T4, T5, T6, T7, T8, T10, T11, T12} son transiciones temporales. Implementarlas y asignarles una ventana de tiempo de sensibilizado [alfa, beta] (a elección del grupo) en milisegundos. TIP: asignar a beta un valor suficientemente grande para que no ocurra el desensibilizado de la transición.

Políticas

Una vez resuelto al TP incluyendo la semántica temporal, es necesario para el modelado del sistema implementar políticas que resuelvan los conflictos relacionados a:

- Mantener la carga en los diferentes invariantes de transición balanceada (entendiendo por balanceado que el promedio de disparo de las transiciones de los invariante en el tiempo sean “aproximadamente” iguales)

Requerimientos

1) Implementar la red de Petri de la Figura 1 haciendo uso de una herramienta, ej: **PIPE**. Verificar todas sus propiedades, hacer un análisis y en caso que exista deadlock en la red, implementar las plazas necesarias para desbloquear la red.
2) El proyecto debe ser modelado con objetos en Java, haciendo uso de un monitor de concurrencia para guiar la ejecución de la red de Petri.
3) Implementar un objeto Política que cumpla con los objetivos establecidos en el apartado Políticas.
4) Hacer el diagrama de clases que modele el sistema.
5) Hacer el diagrama de secuencia que muestre el disparo exitoso de una transición temporal que esté sensibilizada, mostrando el uso de la política.
6) Indicar la cantidad de hilos necesarios para la ejecución y justificar de acuerdo a lo mencionado en el apartado Implementación.
7) Realizar múltiples ejecuciones con 1000 invariantes completados (para cada ejecución), y demostrar con los resultados obtenidos:
    a) Cuán equitativa es la política implementada en el balance de carga en los invariantes.
    b) La cantidad de cada tipo de invariante, justificando el resultado.
8) Implementar la lógica temporal indicada en el apartado Tiempo. De las ejecuciones del punto 7) deberá indicar los tiempos obtenidos y la interpretación de los mismos.
9) Registrar los resultados del punto **7)** haciendo uso de un archivo de log para su posterior análisis.
10) Mostrar e interpretar los invariantes de plazas y transiciones que posee la red.
11) Verificar el cumplimiento de los invariantes de plazas luego de cada disparo de la red.

12) Verificar el cumplimiento de los invariantes de transiciones mediante el análisis de un archivo log de las transiciones disparadas al finalizar la ejecución. El análisis de los invariantes debe hacerse mediante expresiones regulares.
Tip: www.debuggex.com - www.regex.com -.
13) El programa debe poseer una clase Main que al correrla, inicie el programa. Además el programa debe finalizar de manera autónoma sin la intervención del usuario.

Entregables

a) Un archivo de imagen con el diagrama de clases, en buena calidad.
b) Un archivo de imagen con el diagrama de secuencias, en buena calidad.
c) En caso de que el punto 1) de los requerimientos requiera modificar la red de Petri, se debe entregar:
    i) Un archivo de imagen con la red de Petri final, en buena calidad.
    ii) El archivo fuente de la herramienta con la que se modeló la red.
d) El código fuente Java (proyecto) de la resolución del ejercicio.
e) Un informe obligatorio que documente lo realizado, explique el código, los criterios adoptados y que explique los resultados obtenidos.

Subir al LEV el trabajo **TODOS** los participantes del grupo.