Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2650: Fundamentos de Lenguajes de Computación
Tarea 3: Concurrencia y Programación funcional

Viernes 26-Junio-2026

1 Objetivos

En este curso has aprendido que los lenguajes de programación deben ofrecer interfaces de concurrencia y lo básico de programación funcional. Usarás estos conocimientos para reparar la máquina del DCC-tiempo.

1. **Importante:** La evaluación será hecha en base a *test cases*, cada parte de la tarea vale 50%.
2. **Fecha de entrega:** Viernes 26 de Junio
3. Esta tarea cuenta con política de atraso hasta 2 días.

Contenidos

1 Objetivos **1**

2 Introducción **2**

3 Ejercicios **2**
3.1 Haskell (50%) . . . 2
3.1.1 Evaluación . . . 2
3.1.2 Parseando Futuros Contradictorios . . . 2
3.1.3 Generando Pasados . . . 4
3.1.4 Reescribiendo el Presente . . . 4
3.2 Javascript (50%) . . . 5
3.2.1 Enviando una promesa para el futuro . . . 5
3.2.2 Ejemplos . . . 5

4 Entrega **6**
4.1 Classroom . . . 6
4.2 Evaluación . . . 7
4.3 Política de Atraso . . . 7

5 Recomendaciones **7**
5.1 Recomendaciones generales de parte de los ayudantes . . . 7
5.2 Recursos . . . 8
5.2.1 Haskell . . . 8
5.2.2 Javascript . . . 8

6 Uso de inteligencia Artificial **8**

2 Introducción

Has descubierto la máquina del DCC-tiempo oculta en el 5to piso del edificio San Agustín, pero al probarla te diste cuenta que no estaba terminada y terminas viajando por distintas lineas temporales! Debes reparar la máquina del tiempo para volver al presente sin causar anomalías temporales.

3 Ejercicios

3.1 Haskell (50%)

Haskell es un lenguaje de propósito general **puramente funcional**, de tipado estático fuerte y *lazy evaluation*. Fue nombrado a partir de matemático Haskell Curry. Para este curso utilizaremos el compilador GHC, que otorga la implementación oficial, junto con el cabal package manager. Puedes instalar GHC aquí.



3.1.1 Evaluación

Similar a la tarea anterior, puedes elegir **dos de los tres** ejercicios de la sección de Haskell. Si entregas tres ejercicios **entonces se consideraran los dos con menor puntaje**

3.1.2 Parseando Futuros Contradictorios

Debes evitar viajar por futuros contradictorios, por lo que decides modelarlos como fórmulas proposicionales (Well-formed formulas). Debes definir funciones para parsear fórmulas y, entonces, definir operaciones sobre ellas.

Deberás definir la función `isWFF` que valida si una formula está escrita correctamente. Las variables se nombran con una sola letra y, opcionalmente, un número. No se deben utilizar otros carácteres.

```
isWFF "" >>> False
isWFF "p" >>> True
isWFF "q1" >>> True
isWFF "r22" >>> True
isWFF "_p" >>> False
isWFF "pq" >>> False
isWFF "1r" >>> False
isWFF "r/" >>> False
```

Puede haber una cantidad arbitraria de negaciones:

```
isWFF "¬p" >>> True
isWFF "¬¬p" >>> True
isWFF "¬¬¬¬¬¬¬¬¬p" >>> True
```

Solo puede haber un símbolo entre fórmulas. Pueden haber paréntesis para separar y distinguir fórmulas entre si, pero no es obligatorio:

```
isWFF :: String -> Bool
isWFF "p ∨ q" >>> True
isWFF "p ∧ q" >>> True
isWFF "p ∧ q ∧ r ∧ p" >>> True
isWFF "p ∧ q ∨ r " >>> True
isWFF "p ∧ ¬(q ∨ r) " >>> True
isWFF "p ∧ ∧ q" >>> False
isWFF "p ∧ " >>> False
isWFF "(p → q" >>> False
isWFF "p ↔ q)" >>> False
isWFF "∨" >>> False
isWFF "∧ q" >>> False
isWFF "p q" >>> False
isWFF "p ¬q" >>> False
```

Deberás definir la función `getUniqueAtoms` que recibe un WFF y retorna una lista con las variables de la fórmula:

```
getUniqueAtoms :: String -> [String]
getUniqueAtoms "¬p" >>> ["p"]
getUniqueAtoms "p ∧ q" >>> ["p", "q"]
getUniqueAtoms "p ∧ q ∨ r" >>> ["p", "q", "r"]
```

Deberás definir la función `countOperators` que recibe un WFF en string y retorna la cantidad de variables únicas:

```
countOperators :: String -> Int
countOperators "p" >>> 0
countOperators "¬p" >>> 1
countOperators "(p ∧ q) ∨ r" >>> 2
```

Deberás definir la función `depth` que recibe un WFF en string y retorna la profundidad del arbol. Para este ejercicio se entregará un orden de precedencia:

1. Negación.
2. Conjución.
3. Disyunción.
4. Implicancia.
5. Doble implicancia.

```
depth :: String -> Int
depth "p" >>> 1
depth "¬p" >>> 2
depth "¬p ∧ q ∨ r → s ↔ t" >>> 6
```

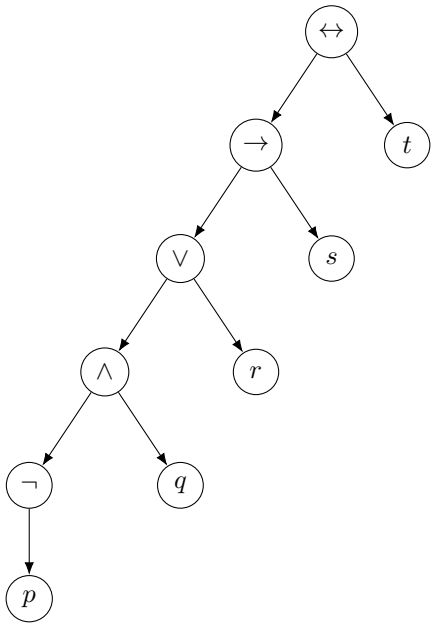


Figure 1: Árbol de sintaxis para: $\neg p \wedge q \vee r \rightarrow s \leftrightarrow t$

3.1.3 Generando Pasados

Existe una cantidad infinita de pasados por los que puedes viajar accidentalmente, por lo que debes encontrar la única a la que perteneces. Una feature de Haskell es que puedes definir estructuras infinitas que son computadas solamente cuando se necesiten.

Deberás definir la lista infinita `powers` que contiene las potencias de dos.

```
powers :: [Int]
take 1 powers >>> [1]
take 2 powers >>> [1, 2]
take 4 powers >>> [1, 2, 4, 8]
take 8 powers >>> [1, 2, 4, 8, 16, 32, 64, 128]
```

Deberás definir la función `cycleSequence` recibe una secuencia y retorna una lista infinita con la secuencia entregada. Está prohibido utilizar la función builtin `cycle`, ya que tiene el mismo comportamiento.

```
cycleSequence :: [a] -> [a]
onetwotree = cycleSequence [1, 2, 3]
take 1 onetwotree >>> [1]
take 3 onetwotree >>> [1, 2, 3]
take 5 onetwotree >>> [1, 2, 3, 1, 2]
take 10 onetwotree >>> [1, 2, 3, 1, 2, 3, 1, 2, 3, 1]
ciclos = cycleSequence "ciclos!"
take 3 ciclos >>> "cic"
take 6 ciclos >>> "ciclos"
take 10 ciclos >>> "ciclos!cic"
```

Deberás definir la lista infinita `mersenne` que contiene la secuencia de los números de mersenne.

```
mersenne :: [Int]
take 1 mersenne >>> [3]
take 8 mersenne >>> [3, 7, 31, 127, 8191, 131071, 524287, 2147483647]
```

3.1.4 Reescribiendo el Presente

Finalmente, la API geométrica de la máquina del tiempo fue dañana: para repararla debes redefinir algunas de sus funcionalidades con grafos simples no direccionales. La API contiene las siguientes estructuras de datos:

```
data Graph a = Graph { nodes :: Set a
                     , edges :: Set (a, a)
                     }
  deriving (Show, Eq)

data GraphCommand a = AddNode a
                    | AddEdge a a
                    | RemoveNode a
                    | RemoveEdge a a
  deriving (Show)

data GraphQuery a = GetNodes
                  | GetEdges
                  | HasNode a
                  | HasEdge a a
                  | GetNeighbors a
                  | hasCycle
                  | GetConnectedComponents
                  | GetShortestPath a a
  deriving (Show)

data GraphQueryResult a = NodeSet (Set a)
                    | EdgeSet (Set (a, a))
                    | BoolResult Bool
                    | PathResult [a]
                    | ComponentResult [Set a]
  deriving (Show)
```

Deberás definir las siguientes funciones:

```
transformGraph :: Graph a -> GraphCommand a -> Graph a
takeGraphQuery :: Graph a -> GraphQuery a -> GraphQueryResult a
```

Algunos ejemplos de tests son los siguientes:

```
emptyGraph = Graph { nodes = Set.empty, edges = Set.empty }
g1 = transformGraph emptyGraph (AddNode 1)
g1 >>> Graph { nodes = Set.fromList [1], edges = Set.empty }
g2 = transformGraph (transformGraph g1 (AddNode 2)) (AddEdge 1 2)
g2 >>> Graph { nodes = Set.fromList [1], edges = Set.fromList [(1, 2), (2, 1)] }
g3 = transformGraph g2 (AddEdge 1 2)
g3 >>> Graph { nodes = Set.fromList [1], edges = Set.fromList [(1, 2), (2, 1)] }

g = Graph {
  nodes = Set.fromList [1, 2, 3, 4],
  edges = Set.fromList [(1,2), (2,1), (2,3), (3,2), (3,4), (4,3)]
}

queryG = takeGraphQuery g

queryG GetNodes >>> NodeSet (Set.fromList [1, 2, 3, 4])

queryG GetEdges >>> EdgeSet (Set.fromList [(1, 2), (2, 1), (2, 3), (3, 2), (4, 3), (3, 4)])

queryG (HasNode 2) >>> BoolResult True

queryG (HasNode 5) >>> BoolResult False

queryG (HasEdge 1 2) >>> BoolResult True

queryG (HasEdge 2 1) >>> BoolResult True

queryG (HasEdge 1 4) >>> BoolResult False

queryG (GetNeighbors 2) >>> NodeSet (Set.fromList [1, 3])

queryG HasCycle >>> BoolResult False

queryG GetConnectedComponents >>> ComponentResult [Set.fromList [1, 2, 3, 4]]

queryG (GetShortestPath 1 4) >>> PathResult [1, 2, 3, 4]
```

3.2 Javascript (50%)

Javascript es un lenguaje de programación interpretado dinámico, multi-paradigma y *event driven*. Fue creado por la compañia Netscape en 1995 para darle un atributo dinámico a las páginas web, que eran puramente estáticas en ese tiempo. Hoy en día es una de las tecnologías clave para el desarrollo web, junto con HTML y CSS.
Para este curso utilizaremos NPM, que viene junto con el runtime Node.js durante la instalación.



3.2.1 Enviando una promesa para el futuro

Debes reparar la funcionalidad de la máquina del tiempo para procesar tareas concurrentemente utilizando las promesas de Javascript. En esta parte definirás las clases `Thread`, que recibe una función, y `PromisesSemaphore`, que utiliza promesas para sus señales.

3.2.2 Ejemplos

Se te entregarán las siguientes clases, donde deberás definir sus constructores y variables, además de rellenar los métodos entregados. Eres libre de modificar las clases siempre y cuando ejecuten los tests.

```
class PromisesSemaphore {
  async wait() {
  }
```

```
  signal() {
  }

  getCount() {
  }
}
class Thread {
  async start() {
  }
}
```

A continuación tendrás un ejemplo sobre cómo se deberían usar las clases. Podrás encontrar más tests en tu repositorio:

```
let runningDuringExecution = false;
const task = async () => {
  runningDuringExecution = true;
};
const thread = new Thread(1, task);
expect(thread.id).toBe(1);
expect(thread.task).toBe(task);
expect(thread.isRunning).toBe(false);
await thread.start();
expect(task).toHaveBeenCalled();
expect(thread.isRunning).toBe(false);
expect(runningDuringExecution).toBe(true);

const semaphore = new PromisesSemaphore();
expect(semaphore.getCount()).toBe(1);
await semaphore.wait();
expect(semaphore.getCount()).toBe(0);
semaphore.signal();
expect(semaphore.getCount()).toBe(1);
semaphore.signal();
expect(semaphore.getCount()).toBe(1);

let blocked = true;
const waiter = semaphore.wait().then(() => {
  blocked = false;
});
await waiter;
expect(blocked).toBe(false);

blocked = true;
semaphore.wait().then(() => {
  blocked = false;
});
expect(blocked).toBe(true);
```

4 Entrega

4.1 Classroom

- *Link classroom:* https://classroom.github.com/a/wYgpPoWu

El repositorio tiene la siguiente estructura de archivos:

```
README.md
haskell/
javascript/
```

Los comandos para ejecutar los tests son:

```
cabal test // en la carpeta haskell
npm test // en la carpeta javascript
```

4.2 Evaluación

- **Importante:** recuerda ejecutar los tests que te entregaremos para asegurar que tu programa funciona.
- Cada parte de la tarea (Haskell y javascript) vale 50% del total de la evaluación.
- Debe modificar el archivo `README.md` con su nombre y número de alumno. Si no los agrega, arriesgas que tu tarea no se pueda revisar.
- Es posible que nos comuniquemos con usted si hay algún trozo de código que genere dudas o sospechas. No responder o no poder explicar el código puede resultar en descuentos a su calificación.
- Siempre y cuando se permita en el ejercicio, siéntanse libres de modificar y agregar clases y métodos.
- Se calificará el porcentaje de tests correctos: es decir que la nota de la tarea equivale a

$$\frac{\texttt{passing tests}}{\texttt{total tests}} * 6 + 1$$

- La **Fecha de Entrega:** es *Viernes 26 de Junio* hasta las 23:59.

4.3 Política de Atraso

Existe la posibilidad de entregar esta evaluación con hasta **2 días de atraso**. En ese caso, **la nota máxima a obtener se reduce** en 10 décimas por cada día de atraso. La fórmula para calcular la nota final con descuento es:

$$\texttt{nota_final} = min(7.0 - (1.0 * dias_d e_a traso), nota_o btenida)$$

En otras palabras, la nota máxima para entregar al día siguiente es de 6.0, y la nota máxima a los dos días siguientes es de 5.0

5 Recomendaciones

5.1 Recomendaciones generales de parte de los ayudantes

- Está disponible un foro de discussions para las dudas de su tarea.
- Los lenguajes de la tarea estan asociados a sus carpetas y archivos de tests correspondiente.
- Los archivos tienen una estructura por defecto para las funciones que se te pide implementar. Se recomienda seguir esta estructura.
- Son libres de editar los archivos con los que viene la tarea (a veces es necesario) siempre y cuando tengan cuidado que los tests sigan corriendo.
- Lee atentamente las notas resaltadas como **Importante** en el enunciado, ya que contiene restricciones sobre como puedes resolver el ejercicio.
- En caso de que falte información en el enunciado o los tests publicos para tomar decisiones respecto a la implementación, es libre de realizar suposiciones en como tratar con ciertos casos. Sin embargo, debes declarar en el archivo `README.md` que suposiciones realizo para cada parte.
- Si utilizas alguna herramienta como soporte para realizar su tarea, debes declararlo en el archivo `README.md`. Ten en cuenta las normas de integridad académica del curso.
- No "copien y peguen" una solución generada.
- Su tarea será revisada manualmente por el cuerpo de ayudantes. Es por esto que se sugiere comentar su código y explicarlo de manera clara.

5.2 Recursos

5.2.1 Haskell

- **Página principal:** https://www.haskell.org/ Aquí puedes encontrar recursos como tutoriales, documentación y canales de ayuda.
- **Uso de ghcup y otras herramientas:** https://www.haskell.org/ghcup/
- **Tutorial completo:** https://learnyouahaskell.github.io/.
- **Haskell online:** https://play.haskell.org/.
- **Cabal:** https://www.haskell.org/cabal/.
- **Unit Testing:** https://hackage.haskell.org/package/HUnit.

5.2.2 Javascript

- **Documentación:** https://developer.mozilla.org/en-US/docs/Web/JavaScript.
- **Promesas:** https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises.
- **Javascript Online:** https://www.programiz.com/javascript/online-compiler/.
- **npm:** https://www.npmjs.com/.
- **Node.js:** https://nodejs.org/en/.
- **Unit Testing:** https://jestjs.io/.

6 Uso de inteligencia Artificial

Respecto al uso de herramientas generadoras de código. Estas están permitidas siempre y cuando se referencie correctamente cuándo es utilizado. Junto a lo anterior, se espera que toda respuesta entregada por una herramienta generadora de código o de IA pase por un proceso de análisis crítico y modificación antes de ser incluida en una evaluación. En caso de entregar una tarea donde se identifique que solo hubo un proceso de copiar la respuesta dada por una herramienta generadora de código o de IA, esto no será aceptado y será considerado como una falta a la integridad académica.