Resumen Parcial 2

Estos resumenes los baso del apunte de la profe. Voy a ir en orden no del apunte, sino de como se dieron los temas en clase:

- Concurrencia
- frameworks
- scripting
- programacion logica
- Programacion asistida
- Seguridad en lengaujes de programacion

Aclaraciones PRELEECTURA NO SE QUE ESCRIBIR

- Programacion logica esta explicada por ejemplos copiadas de las filiminas, se recomienda ver la clase que esta linkeada por ahi haciendo ejercicios
- Programacion asistida no lo resumi porque la profe dijo que son los ejercicios esos de interpretar y "discutir"
- seguridad esta muy zzz

Concurrencia y actores

```
Este capitulo se resume desde el mitchell
```

Un programa concurrente define dos o mas secuencias de acciones que se pueden ejecutar en simultaneo

- Multiprogramming. Un solo procesador corre varios procesos en simultaneo
- Multiprocesado. Dos o mas procesadores comparten memoria, permitiendo que un proceso en un procesador interactue con procesos de otros

14.1 Basic Concepts In Concurrency

Este capitulo habla sobre conceptos básicos de la concurrencia, todas cosas que se vieron en sistemas operativos. Solo voy a "resumir" con palabras claves

- No determinismo
- Atomicidad
- Hilos
- Locks

- Deadlock
- Semaforo
- Condicion de carrera
- Exclusion mutua

No vale la pena leer (al menos que no te acuerdes una goma de sistops)

14.2 The Actor Model

Los actores son una manera general de encarar la computación concurrente. Cada actos es una forma de un objeto reactivo, ecejutando algun compute en respuesta a un mensaje y enviando una respuesta cuando se computa. No tienen un estado compartido, sino que usan mensajes asincronicos pasando todo la ccomunicacion.

Un actor es un objeto que realiza sus acciones en respuesta a las comunicaciones que recibe. Los actores hacen tres cosas basicas

- Manda mensajes a si mismo u otros actores
- Crea actores
- Escpecifica un "replacemente behavior", que sería ser reemplazado por otro actor

La computación de actores es reactiva, que significa que el compute se realiza solo en respuesta a una comunicación. Un programa de actores crea un numero de actores y les envia mensajes. No hay un concepto explicito de un hilo.

Un actor duerme hasta que recibe comunicación. Cuando el actor recibe el mensaje, su script hace sus cosas y luego vuelve a mimir. Su comportamiento de reemplazo le va a decir que si recibe un mensaje actualizando un dato, entonces el va actualizar ese dato.
No existen asignaciones a variables locales.

Un mensaje de un actor a otro se llama *task* que tiene tres partes:

- un tag unico, que lo distingue de otros tasks en el sistema
- un objetivo, que es el destinatario
- una comunicación, data contenida en el mensaje

Figure 14.1 shows a finite set represented by an actor. The actor in the upper left of the figure represents the set with elements 1, 4, and 7. This actor receives a task Insert 2. In response, the actor becomes the actor on the right that has elements 1, 2, 4, and 7. This actor receives the task Get min. As a result, the actor sends the message 1, which is the minimum element in the set, and becomes an actor that represents the set with elements 2, 4, and 7.

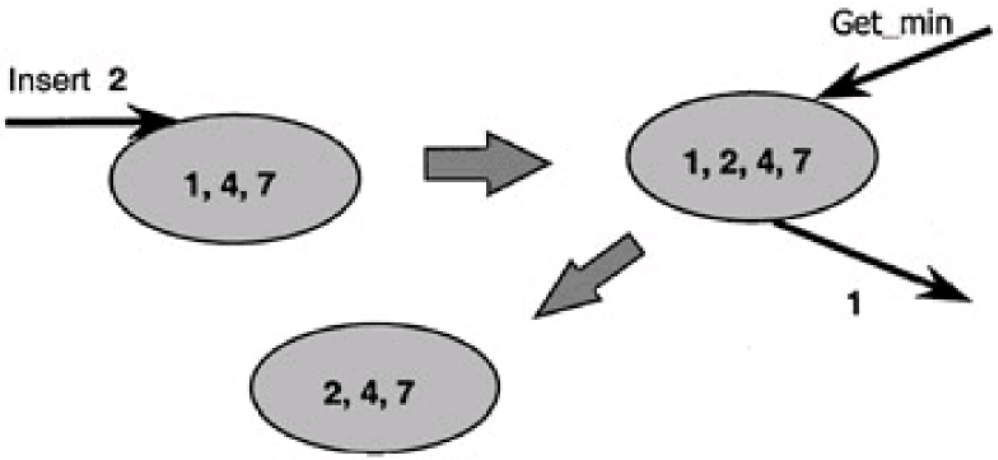


Figure 14.1: Messages and state change for a finite-set actor.

Este ejemplo es muy clave para entender cosas de actores. Vale la pena leer.

14.5 Chapter summary

```
es un resumen de lo que resumi o sea es un resmuen de actores, complicado no?
creo que literal solo voy a traducir la parte de actores, muy clave
```

Los actores no tienen un estado compartido, sino que usan "mensajes asyncronos buffereados" para toda comunicación. Los Actores cambian de estado atómicamente. No hay cambios de estado internos en un actor mientras responde un mensaje. Cambian de estado atómicamente a medida que completa su computo y especifica su compotamiento de reemplazo.

```
Acá resumo de las filminas de la profe
```

La idea de la programación reactiva es ir actualizando los valores a medida que se van asignando
Por ejemplo

```
a = b + c
luego hago
c++
entonces se actualiza el valor de a
```

los actores son livianods
deterministicos
son declarativos
no existe la seccion critica
no se necesitan los

Frameworks

```
este se resume del apunte de la profe (cap 8, pag 84)
```

Un *framework de software* es una abstracción en la cueal un software que provee una funcional genérica se puede modificar en algunos puntos mediante código escrito por un usuario.
Provee un estándar para construir y desplegar aplicaciones.
Facilita el desarrollo de una familia de aplicaciones.

Caracterisitcas clave

- inversión de control: el flujo de control del programa está dirigido por el framework
- extensibildiad: el usuaruio puede extender el framework
- el código del frame work es no modificable: no modificar, solo extender
- frozen spots: componentes basicos que no se cambian
- hot spots: partes donde los programadores meten codigo

8.1 Inversión de control

También conocida como el principio de Howllywood: no nos llame, nosotros lo llamaremos a usted.

> eso la profe siempre lo dice

En la programación tradicional, el código que escribe el programador llama a librerías que se encargan de tareas genéricas. En cambio, con la inversión de control es el framework el que llama al código escrito por el programador.

La inversión de control se usa para aumentar la modularidad del programa y hacerlo extensible

la inversión de control convierte un código escrito secuencialmente en una estructura de delegación. En lugar de que tu programa controle todo explícitamente, el programa configura una clase o librería con algunas funciones específicas para tu aplicación,

8.1.1 Desacoplamiento

se da un ejemplo que no termino de entender, perdon

8.1.2 Boilerplate

Boilerplate es un término que se usa para nombrar porciones del código que se usan en muchos lugares con poca o ninguna alteración.

En programas orientados a objetos, en general hay clases que ya vienen con métodos para obtener y configurar variables de instancia. Las definiciones de estos métodos en

generalse consideran boilerplate.

> Da un ejemplo con mascotas pero yo lo pienso cuando estaba haciendo la defensa del lab y tenia que hacer los metodos "getXX" para obtener los atributos ya me los hacia automaticos

---

Scripting

los lenguajes específicamente de scripting están pensados para ser muy rápidos de aprender y escribir
Los lenguajes de scripting pueden estar diseñados para ser usados por usuarios finales de una aplicación o sólo para uso interno de los desarrolladores. Los lenguajes de scripting suelen utilizar abstracción para hacer transparente a los usuarios los detalles de los tipos internos variables, almacenamiento de datos y la gestión de memoria.

Caracteristicas de los lenguajes de scripting:

- poco verbosos
- sin declaraciones
- reglas de alcance simples
- tipado dinamico flexible
- facil acceso a otros programas
- pattern matching
- manejo de strings muy sofisticado y efeiciente
- tipos de datos de alto nivel (conjuntos, diccionarios, listas, tuplas)

> las siguientes caracteristicas las escuche en clases, por lo que deben ser tomadas con pinzas

- cierta robustez (no suele haber errores por comportamientos inesperados)
-

Glue Languages

Los lenguajes de scripting de propósito general, como Perl y Python, se suelen llamar lenguajes pegamento (glue languages), porque fueron diseñados originalmente para ”pegar”las salidas y entradas de otros programas, para construir sistemas más grandes. Son los que se especializan para conectar componentes de software.
Son útiles para escribir y mantener:

- comandos personalizados para una consola de comandos
- programas más pequeños que los que están mejor implementados en un lenguaje compilado
- pgroamas de contendor para archivos ejecutables, como manipular archivos y hacer otras cosas con el sistema operativo
- secuencias de comandos que pueden cambiar

- prototipos rápidos de una solución

Ejemplos de Glue Languajes

- Perl
- Python
- AWK
- Ruby
- Lua
- Bash (Shell scripting)

Domain Specific languages

Los lenguajes específicos de dominio están pensados para extender las capacidades de una aplicación o entorno, ya que permiten al usuario personalizar o extender las funcionalidades mediante la automatización de secuencias de comandos.

Muchos programas de aplicación grandes incluyen un pequeño lenguaje de programación adaptado a las necesidades del usuario de la aplicación.

Los lenguajes de este tipo están diseñados para una sola aplicación; y, si bien pueden parecerse superficialmente un lenguaje específico de propósito general (por ejemplo QuakeC se parece a C), tienen características específicas que los distinguen, principalmente abstracciones que acortan la expresión de los conceptos propios del juego

---

Programación Lógica

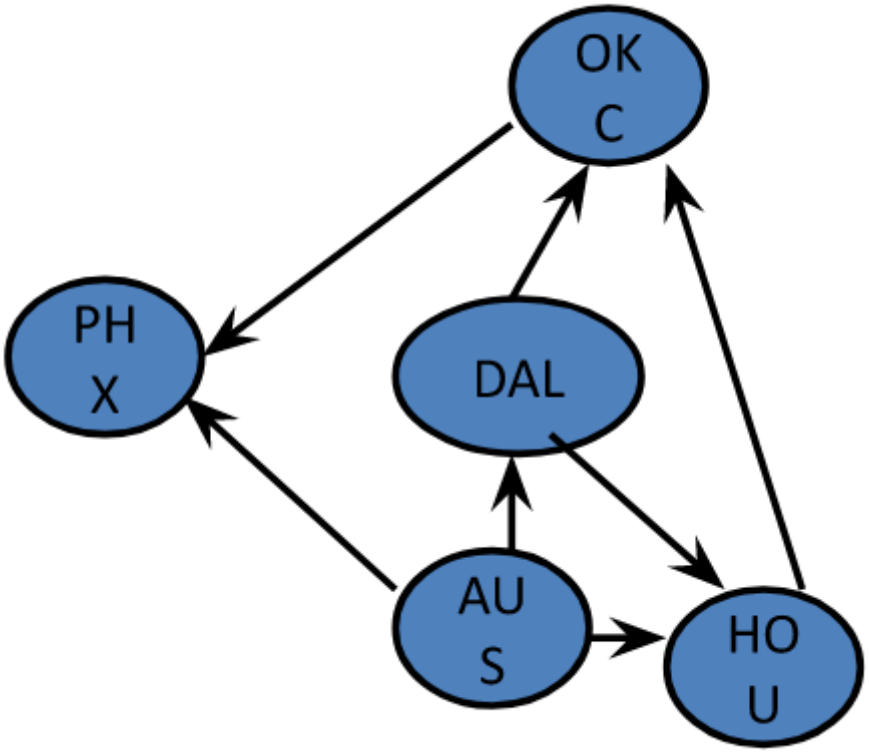
```
aca voy a resumir directo de las filminas porque la profe te manda a estudiar del AULA VIRTUAL DEL 2009 DE INTRO A LOS ALGORITMOS. Sino del mitchell pero no quiero pasarme
```

En programación lógica, la primitiva básica es la relación (predicado)

R(x,y) --> se da relación R entre x e y

"cada línea es una cláusula y representa un hecho conocido (verdades axiomáticas). Un hecho es cierto sii podemos probar usando alguna cláusula"

Este tema se entiende mejor con ejemplos, así que voy a copiar los ejemplos y explicar como los entiendo



En este ejemplo tenemos vuelos de aviones donde una flecha representa un vuelvo "nonstop" de X a Y. Entonces para esta imagén tenemos las siguientes relaciones

```
nonstop(aus,dal).
nonstop(aus,hou).
nonstop(aus,phx).
nonstop(dal,okc).
nonstop(dal,hou).
nonstop(hou,ock).
nonstop(ock,phx).
```

`nonstop(X,Y)` define un vuelo desde X hasta Y

Con esas relaciones definidas podemos empezar a hacer "consultas" usando el operador `?- PREDICADO`

```
?- nonstop(aus,dal).
?- nonstop(dal,okc).
?- nonstop(aus,okc).
```

Eso nos va a devolver

```
yes
yes
no
```

También podemos hacer consultas a ver si existe algún (o varios) X que cumplen el predicado

```
?- nonstop(dal, X).
lo cual nos va a dvolver
X=hou ;
X=okc ;
No
```

```
con lo que tengo entendido como que prolog te devuelve todos los casos donde se
cumplen y luego un No, como que se cansó ????
```

Luego tenemos la conjunción lógica que es combinar condiciones múltiples en una sola consulta

```
?- nonstop(aus, X), nonstop(X, okc).
que nos va a dvolver
X=dal ;
X=hou ;
No
```

Se puden definir nuevo predicados con reglas (similar a funciones)

```
conclusión :- premisas

Por ejemplo

volar_via(Desde, Hacia, Via) :-
        nonstop(Desde, Via),
        nonstop(Via, Hacia).

que la podemos llamar con

?- volar_via(aus,ock,Via).
que nos devuelve
Via=dal ;
Via=hou ;
No
```

Elementos de un programa de Prolog

- Variables: empiezan con mayúscula como `Harry`
- Constantes son enteros o átomos `24, zebra, 'Bob', '.'`
- las estructuras son predicados con argumentots, `n(zebra), habla(Y, Castellano`

Clásulas de Horn
h <- p1, p2, ..., pn
"h es cierto si p1, p2 ... y pn son ciertos simultánemaente"

Hechos, reglas y programas

- hecho es una cláusla de Horn sin parte derecha, `mago(Harry)`
- una regla es una cláusula de Horn con una pare derecha (:- es <--)
  `term :- term1, term2, ... termn`

  - un prorama es un conjuto de hechos y reglas

  habla sobre listas pero yo por ahora lo ignoro

Contestar consultas

- la computación en prolog es buscar una prueba lógica
- aca habla basntate de unas cosas que son como "el interpretado de prolog" no se es un viaje, queda para otra vuelta

Unificación

Dos términos son unificacbles si hay una sustitución de variables que hace que puedan llegar a ser lo mismo
Por ejemplo f(x) y f(3) se unifican con x=3

> acá terminan las filiminas y hay dudas para los ejercicios, después veo que hago

Se recomienda ver esta clase desde el la hora 1:15:00 creo
"en el examen solo tienen que dejar escritos los que se cumplen"

> este tema se entiende mucho mejor con ejercicios

Seguridad

```
el apunte recomienda una sección corta del mitchell pero habla de ja🤢va🤮, por
lo que resumo de las filminas directo
```

hay que defendernos de cosas malicionas

- grafecul degradation
- offensive programming
- cazar bugs
- hacer el codigo predecible
- reducir superficie de atauqe

errores **esperables**

- input invalido
- se agotaron recursos
- fallod el hardware

errores **prevenibles**

- argumentos de funcion invalidos
- valor fuera del rango
- valor de retorno no documentado o excepcion

blacklist: defendese de problemas conocidos
whitelist: defenderse de todo lo desconocido

defensive programming

Defensive programming is a form of defensive design intended to develop programs that are capable of detecting potential security abnormalities and make predetermined responses. It ensures the continuing function of a piece of software under unforeseen circumstances. Defensive programming practices are often used where high availability, safety, or security is needed.

offensive programming

- descubrir todos los malfuncionamientos posibles
- eleminar todas las estrategias del códgo que permiten "salvar errores"

ejemplo de programación “demasiado” defensiva

```
if (is_legacy_compatible(user_config)) {
    // Strategy: Don't trust that the new
code behaves the same
    old_code(user_config);
} else {
    // Fallback: Don't trust that the new
code handles the same cases
    if (new_code(user_config) != OK) {
        old_code(user_config);
    }
}
```

estrategia ofensiva

```
// Trust that the new code has no new bugs
new_code(user_config);
```