UNIVERSIDADE DA CORUÑA

DISEÑO DE LENGUAJES DE PROGRAMACIÓN
GRADO DE INGENIERÍA INFORMÁTICA: ITINIERARIO DE COMPUTACIÓN
CURSO 2014-15

PRÁCTICA 1:
ANÁLISIS DE LAS CARACTERÍSTICAS DE UN CONJUNTO DE LENGUAJES DE PROGRAMACIÓN A PARTIR DE UN CASO PRÁCTICO

[actualizado a 8/10/14]

FECHA LÍMITE DE ENTREGA: viernes 17 de octubre, 22:00

1. OBJETIVO

En esta primera práctica de la asignatura el alumno realizará una aproximación inicial, un tanto informal e indirecta, al estudio de las características de los lenguajes de programación. Para ello, dado un problema de programación y una serie de lenguajes, el estudiante deberá implementarlo en cada uno de ellos, analizando las ventajas, desventajas, limitaciones, etc. que plantea cada uno de dichos lenguajes de cara a la implementación del problema propuesto, sea tanto a nivel de estructuras de datos como de estructuras de control, entrada/salida, etc.

2. EL PROBLEMA

El problema de programación que hemos elegido es el de, **dado un *autómata finito determinista* (AFD)**, eliminar los *estados inaccesibles* que éste contenga y así **obtener su autómata conexo equivalente**. Aunque el concepto de AFD y sus operaciones relacionadas han sido ya introducidos previamente en la asignatura *Teoría de la Computación*, se adjunta a este enunciado un ANEXO en el que se repasan algunos de los conceptos necesarios para la realización de esta práctica, incluyendo la descripción del algoritmo en cuestión.

3. REALIZACIÓN DE LA PRÁCTICA

De acuerdo a lo expuesto anteriormente, la práctica consiste en:

1. **Diseñar e implementar un TAD AFD en una serie de lenguajes de programación** de entre los que indicaremos posteriormente, debiendo incluir en su definición tanto las estructuras necesarias para el almacenamiento de los datos como las operaciones necesarias para su gestión, así como el algoritmo de **obtención de su autómata conexo equivalente**. Deberán cumplirse los siguientes requisitos:
   a) El alumno tendrá **libertad a la hora de diseñar e implementar** tanto las estructuras de datos que darán soporte al TAD (recuérdese que un AFD puede verse como un *grafo dirigido etiquetado*), como el algoritmo de obtención del autómata conexo equivalente.
      - NOTA: A tal efecto el estudiante puede suponer, si lo considera necesario, que los AFDs con los que trabajaremos tendrán un número máximo de estados n_Q y un tamaño máximo del alfabeto n_E.
   b) Sin embargo, una vez escogidos **el diseño e implementación para un primer lenguaje** de programación (puede ser cualquiera, queda a elección del estudiante), el cual designaremos como *"lenguaje original" o* L_{orig}, dichos diseño e implementación **deberán ser respetados al máximo** para el resto de los lenguajes, aún cuando esto

no permita aprovechar las bondades y características de éstos. Sólo se permitirán las adaptaciones que, forzadas por las características del nuevo lenguaje, respeten el espíritu del diseño e implementación "originales".

Por ejemplo: si L_{orig} no gestiona automáticamente la memoria dinámica y estamos desarrollando una nueva implementación en un lenguaje L' que permite tanto la gestión automática como manual, deberemos seguir empleando la gestión manual. O, al contrario, si L_{orig} gestiona automáticamente la memoria y L' no, en ese caso, al no haber otra opción, esta segunda implementación deberá ser adaptada para gestionar la memoria de forma manual.

Otro ejemplo: si L_{orig} es un lenguaje orientado a objetos y hemos implementado al TAD AFD como una clase, a la hora de implementar el problema en un segundo lenguaje L' que fuese imperativo clásico, sin objetos, es obvio que no es posible implementar el AFD como una clase. Sin embargo, sí podríamos implementar un TAD que respetase en la medida de lo posible las estructuras de datos y las operaciones de la implementación original, transformando los atributos en definiciones de tipos con estructura similar y los métodos en funciones que operen sobre aquéllos.

A la hora de dichas adaptaciones, toda decisión de diseño e implementación resultante que no sea obvia/trivial deberá recogerse en la memoria correspondiente a ese lenguaje (véase más adelante) y ser convenientemente justificada. Si resultase del todo imposible la implementación del problema en el nuevo lenguaje respetando mínimamente el diseño e implementación originales, deberá explicarse también el porqué.

c) Las implementaciones se harán **empleando el lenguaje a un nivel básico**, es decir, no se permite el empleo de librerías especializadas que aborden el tratamiento de autómatas, grafos, etc., ni de implementaciones de los mismos ya proporcionadas por el lenguaje o el entorno de programación.
d) Deberán emplearse buenas prácticas de programación, incluyendo una documentación adecuada del código.
e) Siempre que sea posible la implementación se realizará en una librería, paquete o similar.

En la valoración de la práctica se tendrá especialmente en cuenta la heterogeneidad de los lenguajes de programación empleados. Es requisito mínimo para aprobar que se utilicen al menos tres lenguajes de programación. A modo de orientación, se proponen los siguientes lenguajes: Pascal, C, Java, C++, Fortran, Matlab, Go, Pythom, Perl, OCaml, Erlang, Lisp, Prolog, Basic y Ensamblador, por ejemplo. Sin embargo puede utilizarse cualquier otro lenguaje que el alumno considere de interés (en cuyo caso se recomienda consultar previamente al profesor de prácticas).

2. De cara a su utilización y fácil corrección, deberá **implementar además un programa principal** que, bien automáticamente, bien mediante un pequeño menú o funcionalidad equivalente, permita:
   a) Generar el autómata de entrada a partir de su especificación en forma de *string* (véase siguiente apartado) cargándola desde fichero, pasándosela como parámetro de ejecución, leyéndola vía entrada estándar (i.e. `cat fichero.txt | programa_X`), etc.
      - NOTA: A este respecto el estudiante puede suponer que la entrada es correcta, bien formada y que corresponde a un AFD.
   b) Visualizar la representación del autómata de entrada, bien de forma gráfica o tabular (véase Anexo).
   c) Obtener el autómata conexo equivalente al actual.
   d) Visualizar la representación del nuevo autómata equivalente obtenido.

3. **Redactar para cada lenguaje un informe** que incluya:
   a) El compilador que se utilizó, su versión y el sistema operativo empleado.
   b) Una breve descripción, a modo de contextualización, de las características generales de dicho lenguaje (p.ej. Paradigma al que pertenece, si permite o no trabajar con memoria dinámica, en tal caso su gestión, etc.).
   c) Un análisis detallado de las bondades, desventajas, limitaciones, carencias y cualquier característica peculiar o llamativa de dicho lenguaje en relación a la implementación de nuestro problema, tanto a nivel de estructuras de datos como de estructuras de control, entrada/salida, etc.
   d) Las justificaciones respecto a las decisiones de diseño e implementación tomadas así como respecto a las posibles modificaciones que hubiera que introducir para cada lenguaje.
   e) En el caso de aquellas implementaciones que, por tener que adecuarse a la *implementación original*, no pudieron echar mano de alguna característica positiva de su lenguaje, deberemos explicar cómo se implementó, por qué no resulta adecuada dicha implementación en el contexto de este lenguaje, cómo deberíamos haberla implementado aprovechando sus bondades (no es necesario incluir código, sólo explicarlo) y qué ganaríamos con ello.

4. A mayores deberá incluir, únicamente para el *"lenguaje original"*, el pseudocódigo correspondiente al diseño e implementación tanto de las estructuras necesarias para el almacenamiento de los datos como las operaciones necesarias para su gestión, así como el algoritmo de obtención de su autómata conexo equivalente (véase punto 1 anterior).

4. ESPECIFICACIÓN DE UN AUTÓMATA EN FORMATO STRING

A la hora de especificar un autómata $M=\{Q, \Sigma, s, F, \Delta\}$, emplearemos el formato texto que indicamos a continuación:

1. Primeramente, enumeraremos los identificadores de los *estados* del autómata (Q), separados por espacios y con un punto y coma cerrando la lista. Se puede emplear cualquier cadena de caracteres como identificador de un estado.

2. Seguidamente, enumeraremos el conjunto de símbolos de entrada aceptados por el autómata que componen el *alfabeto* (Σ), de nuevo separados por espacios y cerrando la lista con un punto y coma. Por simplicidad, el espacio de nombres válido para estos símbolos será el de los caracteres en minúscula entre la `a` y la `z` y los dígitos del `0` al `9`.

3. A continuación, especificaremos el identificador del *estado inicial* (s) del autómata, seguido de un punto y coma. El estado inicial deberá ser uno de los estados previamente declarados pertenecientes a Q.

4. Seguidamente, deberá aparecer la lista con el conjunto de *estados finales* (F) separados por espacios y cerrada por un punto y coma. Los estados finales deberán pertenecer al conjunto de estados Q previamente declarado.

5. Por último, representaremos la función de transición (Δ) mediante el listado del conjunto de arcos o *transiciones del autómata* en formato:

   $$q_i\ q_j\ \sigma;$$

   donde los elementos de la tripla están separados por espacios en blanco con un punto y coma al final, q_i es el *estado origen* de donde sale el arco, q_j es el *estado destino* a donde llega el arco, ambos pertenecientes al conjunto de estados Q, y σ es el *símbolo* perteneciente al alfabeto Σ que etiqueta dicha transición. De este modo, la transición

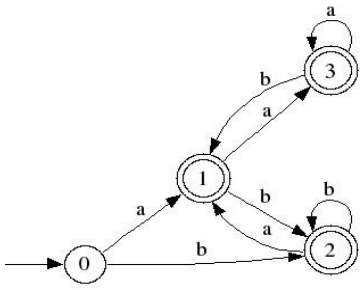
anterior representa que:

$$\Delta(q_i, \sigma)=q_j$$

Así pues, por ejemplo, la especificación:

```
0 1 2 3; a b; 0; 1 2 3; 0 1 a; 0 2 b; 1 3 a; 1 2 b; 2 1 a; 2 2 b; 3 3 a; 3 1 b;
```

representa el siguiente autómata finito:



5. ENTREGA

- Las prácticas son OBLIGATORIAS y se realizarán en grupos de DOS PERSONAS.

- La ENTREGA DE TODAS las prácticas dentro de los plazos acordados y la SUPERACIÓN de la parte práctica de la asignatura (obteniendo entre todas prácticas al menos el 50% de la puntuación de dicha parte) son requisitos IMPRESCINDIBLES para aprobar la asignatura.

- La nota individual de cada una de las prácticas se mantiene para la oportunidad de JULIO. En el caso de tener que acudir a esta segunda oportunidad, y si el alumno así lo desea, éste podrá entregar de nuevo todas o parte de sus prácticas tras corregir las deficiencias encontradas. En tales casos la nota final se corresponderá con la obtenida en esta segunda entrega.

- **Forma de entrega:** Las prácticas deberán depositarse en el directorio habilitado por el CeCaFi a tal efecto (/PRACTICAS/GEI/DLP/P1_Comp). **Ambos miembros** del grupo deberán entregar la práctica en sus respectivos directorios. Dentro del directorio de entrega del usuario éste depositará:

    1. Un único **documento PDF** que incluya:
        a) Portada con los datos de los integrantes del grupo (nombre, login y mail) y un listado de los lenguajes elegidos.
        b) Los informes elaborados para cada uno de dichos lenguajes. El *"lenguaje original"* deberá aparecer de primero, e incluir el pseudocódigo que se pide en el punto 4 del Apartado 3 de este enunciado.

    2. **Un fichero comprimido por cada lenguaje** empleado, cuyo nombre será el de dicho lenguaje (o lo más parecido posible, p.ej. `cplusplus.tgz`). Dicho fichero contendrá un directorio que a su vez contendrá:
        a) El código fuente correspondiente, tanto de la implementación del TAD como del programa principal. De ser posible se incluirá en la cabecera de dichos ficheros los nombres, *logins* y *mails* de los miembros del grupo.
        b) Instrucciones para su compilación (un *Makefile* sería también válido) y ejecución.

5. EVALUACIÓN

- Fecha límite de entrega: **viernes 17 de octubre, 22:00**

- Esta práctica supone el 30% de la nota de la parte práctica de la asignatura.

- La COPIA de una práctica supondrá automáticamente el SUSPENSO (con 0 puntos) de la parte práctica y, consecuentemente, la imposibilidad de superar la asignatura.

- El RETRASO O NO ENTREGA de la práctica supondrán automáticamente la calificación de NO PRESENTADA, imposibilitando también la superación de la parte práctica y, consecuentemente, la superación de la asignatura.

- En caso de que para un lenguaje dado NO SE PRESENTE SU CORRESPONDIENTE INFORME (o éste sea claramente deficiente), DICHO LENGUAJE NO COMPUTARÁ de cara a la nota, independientemente de la implementación.

- De cara a la **valoración de cada solución** entregada se tendrán en cuenta, entre otros, los siguientes aspectos:
  - Cumplimiento de las condiciones especificadas en el enunciado (p.ej. adecuada justificación en la memoria de las decisiones de diseño e implementación).
  - Corrección de la implementación realizada.
  - Eficacia: que se cumplan las especificaciones del problema, implementando correctamente toda la funcionalidad requerida.
  - Esfuerzo y diversidad. Como ya se ha dicho, no es lo mismo, por ejemplo, elegir lenguajes muy similares entre sí, donde las diferencias son poco más que léxicas, a explorar lenguajes de paradigmas muy diferentes unos de otros. Por ejemplo, si el alumno ha implementado ya el problema en Java y lo traslada a C++, es obvio que, dadas las similitudes entre ambos, esta nueva implementación en C++ será menos valorada.
  - Dificultad de la adaptación del diseño e implementación originales al nuevo lenguaje.
  - Uso de buenas prácticas de programación: tales como control de errores, claridad y documentación del código, etc.
  - Claridad y calidad de los contenidos del informe.

- La práctica **deberá ser defendida** ante el profesor posteriormente a su entrega mediante el procedimiento que éste indique. Dicha defensa será OBLIGATORIA y la nota obtenida será individual para cada miembro del grupo. En caso de no haber defendido una práctica dentro de los plazos estipulados a tal efecto, el alumno en cuestión obtendrá la calificación de NO PRESENTADA para dicha práctica, con las consecuencias ya antes citadas.