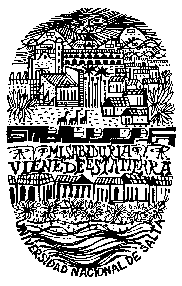
**Trabajo Práctico Transversal**

Teoría de la Computación II

2022



Licenciatura en Análisis de Sistemas

Universidad Nacional de Salta

Nombre: Mauricio Colque Medrano L.U: 225320

Nombre: Fernández Carlos Agustín DNI: 44017405

1. **Introducción**
   1. **Objetivo**

El objetivo de este trabajo practico transversal consiste en la realización de un lenguaje de programación orientado al uso de conjuntos y listas, con el fin de poder llevar a cabo la construcción de las distintas maquinas aprendidas en la materia Teoría de la Computación I.

* 1. **Sobre el nombre del lenguaje**

El lenguaje a desarrollar se denominará **Aleph** puesto que este nombre tiene una fuerte vinculación con la Teoría de Conjuntos, estructura que será el pilar del desarrollo del nuevo lenguaje.

El termino alef es el signo gráfico, correspondiente a la letra hebrea א, empleado por Georg Cantor en la formulación de su teoría de conjuntos para representar la cardinalidad de los números infinitos, es decir, para ordenar los números transfinitos y así diferenciar los distintos tamaños de infinito.

* 1. **Sobre las herramientas a utilizar en el desarrollo**

PONER AQUÍ DESCRIPCIÓN DE FLEX Y BISON

1. **Análisis de requerimientos** 
   1. **Un caso de estudio: Los Autómatas Finitos y sus algoritmos**

Un autómata finito es una “máquina” la cual se encarga de aceptar o rechazar cadenas, tiene un conjunto de estados los cuales determinan su comportamiento. Se dividen en dos grupos:

Autómata finito determinista:

Un autómata finito determinista (afd) A es una 5-upla definida por los siguientes elementos:

* Q Es un conjunto finito no vacío de **estados**.
* Σ Es un conjunto finito no vacío de símbolos llamado **alfabeto de entrada**
* δ QxΣ→Q es una función completamente definida llamada **función de transición**
* q0 ∈ Q es el **estado inicial**
* F ⊆ Q es el conjunto de **estados de aceptación**

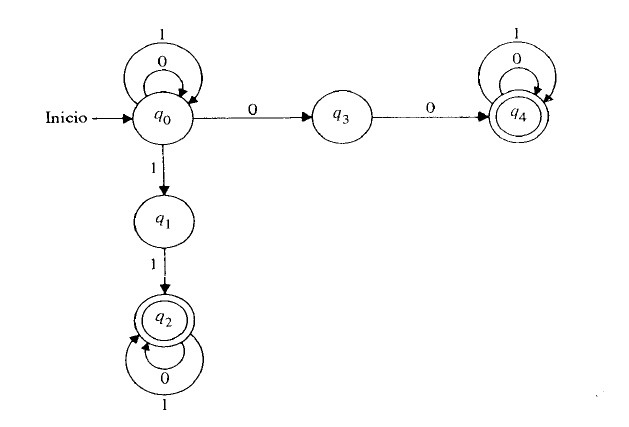
Autómata finito no determinista:

Un autómata finito no determinista (afnd) A es una 5-upla definida por los siguientes elementos:

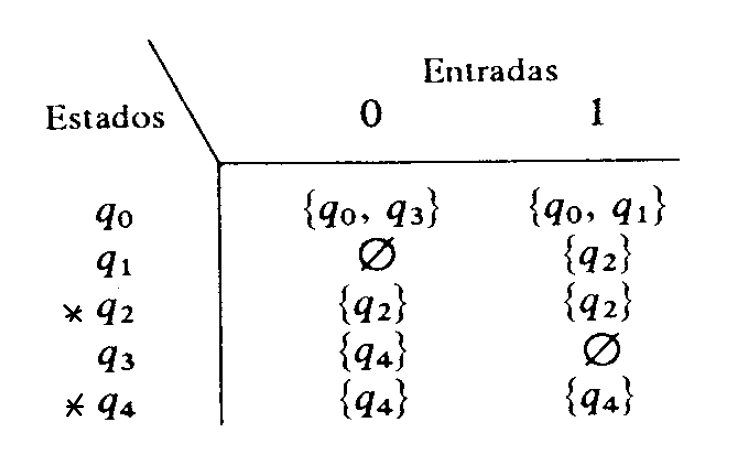
* Q Es un conjunto finito no vacío de **estados**.
* Σ Es un conjunto finito no vacío de símbolos llamado **alfabeto de entrada**
* δ QxΣxQ es una relación llamada **relación de transición**
* q0 ∈ Q es el **estado inicial**
* F ⊆ Q es el conjunto de **estados de aceptación**

Ejemplo de un autómata finito:

Representado en un diagrama de transiciones:



En una tabla:



Conjuntos y listas:

[**{**q0,q1,q2,q3,q4**}**,

**{**0,1**}**,

**{[**q0,0,**{q0,q3}]**,**[**q0,1,**{q0,q1}]**,**[**q1,0,**{}]**,**[**q1,1,**{q2}]**,**[**q2,0,**{q2}]**,**[**q2,1,**{q2}]**,**[**q3,0,**{q4}]**,**[**q3,1,**{}]**,**[**q4,0,**{q4}]**,**[**q4,1,**{q4}]}**,

**q0**,

**{**q2,q4**}**]

Los autómatas finitos deterministas y no deterministas aceptan lenguajes regulares, luego, dado un afnd podemos encontrar un afd equivalente, es decir, que acepta el mismo lenguaje, usando un algoritmo de conversión:

Pseudocódigo:

afnd2afd(A)=

sea A=( Q, Σ, δ, q0, F)

QB= {{q0}}

mientras ∃**R** ∈ QB tal que δB(**R,a**) este definido ∀ **a**∈ Σ

∀ **a**∈ Σ

δB(**R,a**) = **∪**δ(q,a) , q∈ **R**

QB= QB ∪ { δB(**R,a**) }

FB = {**S** ∈ QB | **S ∩** F ≠ Ø }

contestar (QB, Σ, δB, {q0},FB)

Luego, intentamos desarrollar este algoritmo en algún lenguaje de programación, en este caso C, el cual presentará algunas dificultades a la hora de intentar hacer uso de estructuras como Conjuntos y Listas, pues no fue desarrollado para ello:

Código en C:

tData ConversionAFD(tData AFND)

{

    tData sigmaND = returnElem(AFND, 2);

    tData transitionsND = returnElem(AFND, 3);

    tData initialStateND = returnElem(AFND, 4);

    tData acceptanceStatesND = returnElem(AFND, 5);

    tData AFD = newData("[]");

    tData initialStateD = newNestedData(initialStateND, SET);

    tData stateSetD = newNestedData(initialStateD, SET);

    tData sigmaD = copyData(sigmaND);

    tData transitionsD = newData("{}");

    tData acceptanceStatesD = newData("{}");

    tData aux1;

    tData aux2;

    tData partialState;

    tData newTransition;

    int positionInStateSetD = 1;

    while (positionInStateSetD <= CARDINAL(stateSetD))

    {

        for (int positionInSigmaND = 1; positionInSigmaND <= CARDINAL(sigmaND); positionInSigmaND++)

        {

            partialState = newData("{}");

            for (int positionInStateD = 1; positionInStateD <= CARDINAL(returnElem(stateSetD, positionInStateSetD)); positionInStateD++)

            {

                for (int positionInTransitionsND = 1; positionInTransitionsND <= CARDINAL(transitionsND); positionInTransitionsND++)

                {

                    if (isEqual(returnElem(returnElem(stateSetD, positionInStateSetD), positionInStateD), returnElem(returnElem(transitionsND, positionInTransitionsND), 1)))

                    {

                        if (isEqual(returnElem(sigmaND, positionInSigmaND), returnElem(returnElem(transitionsND, positionInTransitionsND), 2)))

                        {

                            aux1 = UNION(partialState, returnElem(returnElem(transitionsND, positionInTransitionsND), 3));

                            dataFree(&partialState);

                            partialState = aux1;

                            aux1 = NULL;

                            break;

                        }

                    }

                }

            }

            newTransition = newData("[]");

            PUSH(newTransition, copyData(returnElem(stateSetD, positionInStateSetD)));

            PUSH(newTransition, copyData(returnElem(sigmaND, positionInSigmaND)));

            PUSH(newTransition, copyData(partialState));

            aux1 = newNestedData(newTransition, SET);

            dataFree(&newTransition);

            newTransition = NULL;

            aux2 = UNION(transitionsD, aux1);

            dataFree(&transitionsD);

            dataFree(&aux1);

            transitionsD = aux2;

            aux2 = NULL;

            aux1 = NULL;

            if (!IN(stateSetD, partialState))

            {

                aux1 = newNestedData(partialState, SET);

                dataFree(&partialState);

                partialState = aux1;

                aux1 = UNION(stateSetD, partialState);

                dataFree(&stateSetD);

                stateSetD = aux1;

                aux1 = NULL;

            }

            partialState = NULL;

        }

        positionInStateSetD++;

    }

    for (int positionInStateSetD = 1; positionInStateSetD <= CARDINAL(stateSetD); positionInStateSetD++)

    {

        aux1 = INTER(returnElem(stateSetD, positionInStateSetD), acceptanceStatesND);

        if (!isEmpty(aux1))

        {

            dataFree(&aux1);

            aux1 = newNestedData(returnElem(stateSetD, positionInStateSetD), SET);

            aux2 = UNION(acceptanceStatesD, aux1);

            dataFree(&acceptanceStatesD);

            acceptanceStatesD = aux2;

            aux2 = NULL;

        }

        dataFree(&aux1);

    }

    PUSH(AFD, stateSetD);

    PUSH(AFD, sigmaD);

    PUSH(AFD, transitionsD);

    PUSH(AFD, initialStateD);

    PUSH(AFD, acceptanceStatesD);

    return AFD;

}

La principal dificultad que se encuentra a la hora de desarrollar el programa es debido a que el lenguaje no está pensado para este tipo de problemática.

Se complica mucho la escritura por la falta de estructuras de control que permitan trabajar fácilmente con conjuntos y listas, por ejemplo, una que permita hacer algo con cada elemento de un conjunto u otra que permita sobrescribir un conjunto con su union con otro sin tener que realizar manualmente la liberación de memoria.

Además, la lectura del código es muy complicada por el tema de los parámetros anidados cuando se quiere referenciar por ejemplo a un elemento dentro de un conjunto, dentro de una lista.

* 1. **Conceptos sobre lenguajes de programación**

La implementación de los lenguajes de programación, como todo desarrollo de software, comienzan por el análisis del problema que requiere una solución informática y el diseño de una respuesta adecuada.

Un lenguaje de programación es un sistema notacional para describir computaciones en una forma legible tanto para la maquina como para el ser humano.

Entendemos que una computación es aquel proceso que puede ser efectuado por una computadora, que abarca tanto la manipulación de datos como el procesamiento de texto, almacenamiento y recuperación de la información, entre otros.

Es importante que un lenguaje sea legible tanto para la máquina como para el ser humano, puesto que, para ser traducido a un lenguaje entendible por la computadora, éste debe tener una estructura que permita una traducción eficiente. Por nuestra parte, necesitamos que proporcione abstracciones de las acciones de las computadoras que sean fáciles de entender, por lo que en consecuencia el lenguaje debe parecerse a un lenguaje natural, para que la persona que lo use pueda describir de la mejor manera posible las instrucciones que le brinda a la computadora, basándose en su lenguaje natural.

* 1. **Abstracciones**

Las abstracciones de los lenguajes de programación se agrupan en dos clases generales: **abstracción de datos y abstracción de control,** y también, se agrupan en niveles, representando en cada uno la cantidad de información contenida de la abstracción.

Abstracciones de datos

Se dividen en tres grupos:

1. Abstracciones básicas: Representan valores de datos comunes en una computadora, conocidos como variables cuyo tipo de valor se conoce como tipo.
2. Abstracciones estructuradas: Se trata de colecciones de valores de datos relacionados entre sí, que pueden ser constituidos por diferentes tipos de datos.
3. Abstracciones unitarias: Representan componentes o contenedores, son estructuras de datos que a su vez contienen otros definidos por el usuario (bibliotecas).

Abstracciones de control

* 1. Abstracciones elementales:

Son aquellos enunciados o sentencias en un lenguaje que combinan unas cuantas instrucciones de máquina en una sentencia abstracta mas comprensible.

* 1. Abstracciones estructuradas: Dividen un programa en grupos de instrucciones que están anidadas dentro de pruebas que gobiernan su ejecución.
     1. **Abstracciones de Aleph**

Abstracciones de datos:

* Abstracciones básicas: Elementos
* Abstracciones estructuradas: Conjuntos, listas

Abstracciones de control:

* Abstracciones elementales: Asignación, unión, diferencia, intersección
* Abstracciones estructuradas: Condicional (if), ciclo condicionado (while), ciclo incondicionado (for), *para todo* (foreach)
  1. **Dominio de programación de Aleph**

Aleph se ubica en los lenguajes con dominio científico, esto se debe a que permitirá usar estructuras matemáticas como conjuntos y listas con sus respectivas operaciones.

1. **Diseño** 
   1. **Sentencias**

PONER AQUÍ DEFINICIÓN DE SENTENCIAS

* + 1. *Un primer diseño de sentencias de Aleph*

PONER AQUÍ ALGORITMO DE CONVERSIÓN PROGRAMADO EN POSIBLES SENTENCIAS DE ALEPH

* 1. **Reconocimiento de patrones. Tokens y Lexemas**

PONER AQUÍ DEFINICIÑON DE TOKENS Y LEXEMAS

* + 1. *Tokens y lexemas de Aleph*

PONER AQUÍ TABLA CON TOKENS Y LEXEMAS DE ALEPH

* 1. **Sintaxis: Métodos Formales**

Los lenguajes de programación requieren una interpretación de sus sentencias sin ambigüedades, su descripción, a los fines de comunicar su funcionamiento, tanto a los usuarios de los mismos como a quienes realizan su implementación, requiere herramientas formales.

PONER AQUÍ DEFINICIÓN DE SINTAXIS DE LOS LP

PONER AQUÍ TEORÍA: LAS GLC Y LA DESCRIPCIÓN DE SINTAXIS

PONER AQUÍ DEFINICIÓN DE BNF Y EBNF

* + 1. *Primera descripción formal de Aleph*

PONER AQUÍ EL DESARROLLO DE LA BNF DE ALEPH

* + 1. *Descripción formal de la sintaxis Aleph*

Las instrucciones que conforman todo lenguaje se pueden agrupar en instrucciones de asignación y estructuras de control de flujo. Las instrucciones de asignación también se conocen como de transferencia de datos, pues asignan valores de una dirección a otra o de una expresión a una dirección. Las expresiones que son asignadas o evaluadas manipulan datos y retornan un valor determinado. La evaluación de expresiones puede servir entonces tanto para ser asignada a una dirección de memoria (nominada mediante un identificador) o para determinar bifurcaciones en el flujo de ejecución del programa.

Se presentan a continuación las BNF y EBNFs de las diferentes instrucciones que conforman Alpeh.

**Asignación y Asignación Múltiple**

Las asignaciones son parte fundamental de un lenguaje de programación imperativo, sin ellas no sería posible relacionar valores con variables para un uso posterior.

Asignación

--->poner aquí la BNF de la asignación y borrar esto

Asignación Múltiple

--->poner aquí la BNF de la asignación múltiple y borrar esto

Para transformar valores y observar relaciones entre ellos debemos contar con expresiones que puedan evaluarse. Se presentan a continuación las expresiones de Aleph.

**Asociatividad y Precendencia de operadores**

---> poner aquí la Def de asociatividad y precedencia de operadores, explicación de cómo se implementan estyos conceptos en las BNF

---> poner aquí la BNF de expresiones aritméticas (números ver en sebesta) con precedencia y asociatividad con uso de paréntesis

---> poner aquí la Árbol de derivación de la expresión pedida en el TP 2 parte 2

---> poner aquí la Def árbol de sintaxis abstracta y derivación

---> poner aquí la Def operadores sobrecargados

**Expresiones: Operaciones con Conjuntos y Listas**

---> poner aquí la BNF de Operaciones con Conjuntos y Listas

**Expresiones: Operaciones Relacionales**

---> poner aquí la BNF de Operaciones Relacionales

**Expresiones: Operaciones Lógicas**

---> Def evaluación perezosa e ingenuas

---> poner aquí la BNF de Operaciones Lógicas

**Estructuras de Control**

El control del flujo de ejecución de instrucciones permite alterar el orden en el que cada sentencia es evaluada.

--->Explique sobre la ambigüedad de la sentencia if

--->poner aquí la BNF y EBNF del if

--->poner aquí las BNF de las otras estructuras de control

**Subrutinas**

Las subrutinas permiten la modularización de un programa, de manera que no deba repetir código cada vez que deseo obtener un resultado aplicando argumentos a un algoritmo bien determinado, al que podemos llamar función o procedimiento, dependiendo de si el mismo devuelve o no algún dato simple o estructurado.

--->poner la BNF de la subrutina

**Gramática completa de Aleph**

--->poner aquí la BNF completa

* 1. **Semántica: Métodos Formales**

En esta sección se especificará la semántica, el significado, de las construcciones sintácticas descriptas en la sección anterior del presente informe. Las herramientas formales para esta tarea no son tan conocidas como las herramientas formales para la sintaxis. No obstante, estas herramientas pueden ser utilizadas para complementar las definiciones de algunas construcciones que exijan una mayor precisión en su descripción semántica que aquella que puede brindarse mediante una definición coloquial de su funcionamiento.

--->poner aquí Def Semántica

* + 1. *Semántica Estática*

--->Poner aquí Def Semántica Estática

**Gramática de Atributos**

--->Poner aquí Def Gramática de Atributos

Gramática de atributos para las expresiones de Aleph

--->Poner aquí las gramáticas de atributos de las operaciones unión intersección y diferencia

* + 1. *Semántica Dinámica*

--->Poner aquí Def Semántica Dinámica

**Semántica Operacional**

--->Poner aquí Def Semántica Operacional

Descripción las estructuras de control de Aleph

--->Poner aquí definición de lenguaje intermedio para describir Aleph

--->Poner aquí las descripciones de las estructuras de control de Apleh con semántica operacional

**Semántica Denotacional**

--->Poner aquí Def Semántica Denotacional

Descripción las estructuras de control de Aleph

--->Poner aquí las descripciones de las estructuras de control de Apleh con semántica denotacional

1. **Fase 1 de implementación de Aleph**
   1. **Tipos de implementación**

--->Def de los distintos enfoques de implementación de lenguajes

--->Enfoque de implementación de Aleph

* 1. **Análisis Léxico**

--->Def de Análisis Léxico

--->Descripción de las tareas de un analizador léxico

-->Avance de código: Primer analizador de Aleph (primer scanner que imprime por pantalla los tokens)

* 1. **Análisis Sintáctico**

--->Def de Análisis Sintáctico

--->Metas del Análisis Sintáctico

--->Analizadores Top-Down

--->Gramáticas LL

--->Analizadores Descendentes Recursivos

--->Ejemplo de módulos descendentes recursivos de las expresiones de Aleph

--->Analizadores Bottom-Up

--->Proceso

--->Def Frase, Frase Simple, Manejador

--->Ventajas

--->Funcionamiento de los analizadores LR

--->Funcionamiento del analizador generado por Bison

--> Primer Analizador Sintáctico de Aleph (un archivo .l y un archivo .y relacionados, lo entregado para el 1P)

1. **Fase 2 de implementación de Aleph**
   1. **Construcción del AST y evaluación de expresiones**

-->Definición de AST

-->Mostrar y explicar la estructura de datos y módulos que se usan para la construcción del AST

-->Mostrar y explicar la estructura de datos y módulos utilizados para evaluar los AST en Aleph

-->Evaluación de literales listas, conjuntos (sin elementos repetidos) y elemento (con sus operaciones no excluyente)

* 1. **Variables y la Tabla de Símbolos**

-->Describa las componentes de la seis-upla que caracteriza a una variable.

-->Tabla de símbolos, definición y explicar cómo se implementa la tabla de símbolos y mostrar código en aleph

* 1. **Análisis Semántico**
     1. *Ligaduras*

-->Defina ligadura y tiempos de ligadura.

-->Defina ligadura estática y ligadura dinámica

-->Defina variables estáticas, dinámicas de pila, explicitas de montón dinámico, implícitas de montón dinámico. Enumere las ventajas y desventajas de cada una.

-->Relacione estos conceptos con la implementación de Aleph

* + 1. *Ámbito*

-->Defina: tiempo de vida, alcance, alcance estático y alcance dinámico.

-->Defina: ancestro estático y dinámico de un subprograma

-->Relacione estos conceptos con la implementación de Aleph

* + 1. *Tipos de Datos*

-->Defina: Chequeo de tipos. ¿Qué es un tipo compatible?

-->Defina: error de tipo.

-->Defina: fuertemente tipado y enumere las ventajas de los lenguajes de este tipo.

-->Defina: conversión implícita y conversión explícita de tipos.

-->Relacione estos conceptos con la implementación de Aleph

* + 1. *Semántica de las asignaciones y expresiones*

-->Explicar y mostrar la implementación de las asignaciones y expresiones de Aleph

--> Evaluación de asignaciones

(segundo parcial 25/11 presentación informe defensa lunes 28/11 15hs - recu: sábado 03/12)

1. **Fase 3 de implementación de Aleph (examen final)**
   1. **Semántica de las estructuras de control**

-->Explicar y mostrar la implementación de todas las estructuras de control Aleph, esto incluye las condiciones lógicas y los operadores relacionales.

* 1. **Semántica de las subrutinas**

-->Explicar y mostrar la implementación de las subrutinas de Aleph