

TECNOLÓGICO DE COSTA RICA
Escuela de Ingeniería en Computación
Proyecto de Ingeniería de Software

Profesora:

María Estrada Sánchez

Entrega 3:

Analizador Contextual:
Manual Técnico.

Estudiantes:

Christian León Guevara - 2013371982
Gabriel Ramírez Ramírez - 201020244

Fecha de entrega:

20-01-2019

Período Verano
Cartago

Tabla de contenido

1. Introducción.....	3
2. Ámbito del Sistema.....	4
3. Arquitectura del Sistema.	5
4. Pruebas.....	6
5. Reflexión.....	6
6. Apéndices.....	8

1. Introducción.

1.1. Visión general

En el proyecto del Analizador Contextual se hace una reimplementación del compilador del lenguaje Triángulo que originalmente se encuentra escrito en Java. Esta nueva reimplementación se hizo en una versión del lenguaje funcional ML específicamente en OCaml.

El producto es un sistema técnico conformado por varios componentes que coexisten y colaboran entre ellos. Dentro de esta estructura de componentes nosotros hemos desarrollado el Analizador Contextual.

Una de las características importantes del proyecto es que se necesita interactuar con otros componentes del sistema, por esta razón se hace uso de interfaces que tendrán la definición de los ASTs para la comunicación e interacción de los componentes.

Es importante mencionar que algunas partes utilizadas para la resolución del problema del Analizador Contextual fueron desarrolladas por los compañeros José Antonio Alpízar Aguilar, Pablo Josué Brenes Jimenes y Luis José Castillo Valverde del curso de Proyecto de Ingeniería de Software quienes elaboraron el componente de Analizador Sintáctico.

1.2. Definición del problema

Existe una implementación de un compilador para el lenguaje de programación Triangulo utilizado por el profesor Ignacio Trejos Zelaya en el curso 'Compiladores e intérpretes', de la carrera de Ingeniería en Computación en el TEC. El compilador es de tres pasadas y se encuentra escrito en Java.

Lo que se realizó es una reimplementación del componente del Analizador Contextual escrita en el lenguaje de programación OCaml. Dicho Analizador debe interactuar con otros componentes existentes (Analizador Sintáctico y Generador de Código) comunicándose a través de interfaces para los ASTs que se generan en las distintas fases del proceso de compilación.

1.3. Justificación

Este proyecto surge con el propósito de utilizarse con fines académicos por parte del profesor Ignacio Trejos Zelaya en el curso Compiladores e Intérpretes de la carrera Ingeniería en Computación en el TEC.

Se busca que estudiantes a futuro puedan desarrollar otros componentes del compilador y llegar a desarrollar un compilador completo para el lenguaje Triángulo.

2. Ámbito del Sistema.

2.1 Objetivos específicos

- A. Comprender el funcionamiento de los distintos lenguajes de programación que se usan en el proceso de desarrollo del componente Analizador Contextual.
- B. Analizar el funcionamiento que tiene compilador para el lenguaje Triángulo escrito en Java.
- C. Definir el diseño, algoritmos y estructura para el manejo de la tabla de identificación.
- D. Diseñar casos de prueba correctos e incorrectos para validar el correcto funcionamiento del componente Analizador Contextual.
- E. Documentar los distintos elementos relacionados con el proyecto: pruebas, documentación técnica y manuales.
- F. Realizar la integración de Analizador Sintáctico con el componente del Analizador Contextual.

2.2 Objetivo General

Construir un sistema técnico en OCaml que se acople a una estructura de componentes de un compilador y permita realizar el análisis contextual lenguaje Triángulo y se pueda integrar con el Analizador Sintáctico desarrollado en otra fase del proyecto.

2.3 Criterios de Éxito

- A. Poder utilizar los lenguajes Ocaml y Triángulo de manera fácil, comprendiendo su sintaxis y estructuras.
- B. Comprender correctamente como funciona el compilador de Triángulo que está desarrollado en Java.
- C. Imprimir el contenido de los ASTs para ver la estructura que tienen los distintos casos de prueba desarrollados.
- D. Imprimir el contenido de la tabla de identificación.
- E. Recorrer el AST generado en el Analizador Sintáctico y probar las distintas combinaciones generadas.
- F. Reportar errores generados en la fase del análisis contextual.
- G. Demostrar mediante las pruebas de validación y verificación el correcto funcionamiento del Analizador Contextual.
- H. Que la integración entre el componente Sintáctico y el Contextual sea correcto y funcione sin ningún problema dentro de la estructura del compilador.

2.4 Funcionamiento

Se muestra una tabla con los recursos utilizados y que son necesarios para el desarrollo del proyecto.

Recurso	Versión	Explicación
Ocaml	4.02.3	Es el lenguaje de programación utilizado para el desarrollo del componente del Analizador Contextual.

Sitio web para la obtención del recurso: <https://caml.inria.fr/download.en.html>

3. Arquitectura del Sistema.

La arquitectura del compilador para el lenguaje Triángulo se basa en una estructura de componentes que pueden interactuar entre sí mediante el uso de interfaces. Por eso existen elementos propios del análisis contextual y otros elementos comunes con otros componentes.

Análisis Contextual:

- A. IdEntry.mli: Es una interfaz que define la estructura de cada entrada de la tabla de identificación la cuál es una terna: identificador, ast y nivel.
- B. IdentificationTable.ml: Define la estructura y algoritmos necesarios para la manipulación de la tabla de identificación.
- C. Checker.ml: Contiene todas las funcionalidades que debe realizar el analizador contextual.
 - a. Elementos Comunes:
- D. ErrorReporter.ml: Contiene las funcionalidades para el manejo y reporte de errores.
- E. IdentificationTablePrinter_XML.ml: Contiene todas las funcionalidades que permiten el recorrido y escritura de la tabla de identificación generada el análisis contextual.
- F. TreeDrawer.ml: Es un elemento compartido entre el análisis sintáctico y contextual que permite la escritura de los ASTs en formato XML.
- G. Ast.mli: Interfaz que establece la estructura del AST para la comunicación entre el Analizador Sintáctico y Contextual.

4. Pruebas.

Para este apartado se recomienda revisar los documentos *Diseño de Pruebas y Revisiones.docx* y *Reporte de pruebas, validación y verificación.docx*

Estos documentos contienen todo lo relacionado al plan de pruebas, su correspondiente ejecución y los resultados obtenidos.

5. Reflexión.

5.1 Evaluación

El proyecto ha representado un desafío interesante para el equipo de trabajo ya que había que poner en práctica habilidades en áreas menos desarrolladas en el ciclo de vida de un proyecto.

Sin embargo, es una experiencia de gran aprendizaje sobre el funcionamiento de una estructura para un compilador distribuida en componentes que pueden interactuar entre sí.

El proyecto nos permitió la adquisición de conocimientos en lenguajes como Triángulo y OCaml. El primero como un lenguaje especial para ser utilizado en funciones académicas y el segundo como una oportunidad de conocer más sobre el paradigma funcional.

5.2 Lecciones aprendidas

- A. Es necesario realizar y cumplir con una planificación del tiempo que se va a destinar a cada tarea.
- B. Cuando existe una base del proyecto sobre la cuál se trabaja ocasiona cierto nivel de confianza y puede resultar perjudicial en el desarrollo del proyecto.
- C. La comunicación con el cliente es de vital importancia ya que en ciertos requerimientos es el único que puede aclarar las dudas completamente y así se garantice el avance del proyecto.
- D. La comunicación entre los miembros del equipo de desarrollo es sumamente importante ya que permite evaluar el ciclo de desarrollo del proyecto.
- E. La distribución de tareas técnicas permite un avance más constante del desarrollo del proyecto.
- F. Es necesario desarrollar casos de prueba desde el inicio del desarrollo del proyecto los cuáles garanticen ambientes buenos y malos para la ejecución del proyecto.

- G. Es necesario generar la impresión de los ASTs y la Tabla de Identificación como medio probatorio para el funcionamiento del componente.

5.3 Errores y limitaciones

Dentro del desarrollo de software existen muchos elementos que quedan fuera del alcance de los objetivos del proyecto y cosas que influyeron para el buen desarrollo de este. A continuación, se presenta los principales:

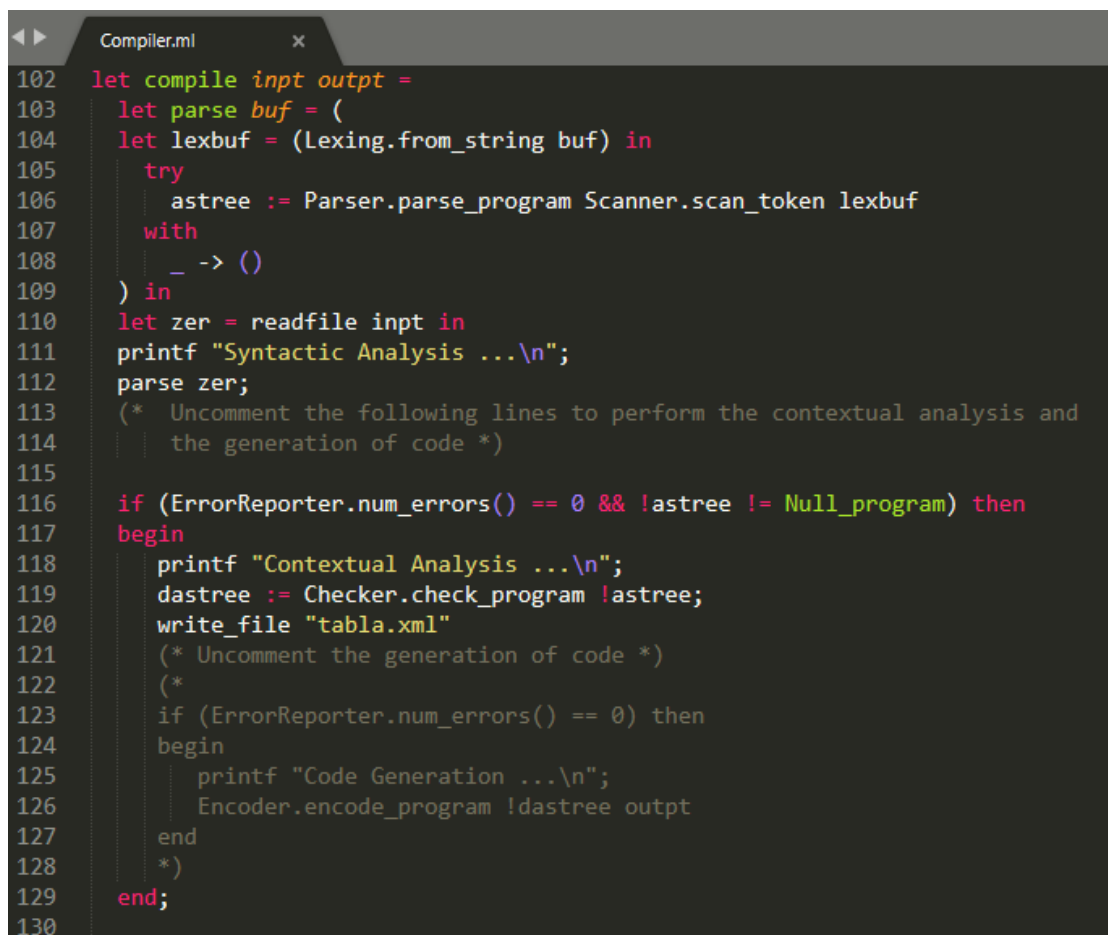
- A. No realizar una investigación inicial de manera extensa con el objetivo de comprender los lenguajes de programación Triángulo y OCaml.
- B. No realizar una investigación inicial de manera extensa con el objetivo de comprender el funcionamiento del compilador para Triángulo desarrollado en Java.
- C. No haber realizado una planificación inicial de los tiempos requeridos por tarea y los encargados de cada una.
- D. Fallos en la comunicación entre los miembros del equipo hasta ya estar avanzado el desarrollo del proyecto.
- E. Adoptar una actitud confiada por la base previa que existía del código del análisis contextual.
- F. Triángulo es diacrítico y en su definición no acepta caracteres especiales como: la ñ, vocales con tildes y diéresis. Es una restricción léxica del lenguaje original.
- G. Se presenta un error a la hora de imprimir la tabla de identificación pues el recorrido se está haciendo a la inversa. Es una mejora para una siguiente etapa del desarrollo del compilador.

6. Apéndices.

6.1 Integración del Analizador Sintáctico y Analizador Contextual

La integración entre ambos componentes se hizo de manera sencilla debido a que únicamente hubo insertar las carpetas del Analizador Contextual y los elementos compartidos para que funcionara. Para una mejor comprensión de la estructura de carpetas se puede consultar el archivo *Entregables, distribución de carpetas.docx*.

Luego integrar las carpetas en un solo proyecto era necesario hacer la invocación del Analizador Contextual mediante la instanciación de un elemento *Checker.ml* dentro del programa principal ubicado en el archivo *Compiler.ml*. Esto se muestra en la siguiente imagen:



```
102 let compile inpt outpt =
103   let parse buf = (
104     let lexbuf = (Lexing.from_string buf) in
105     try
106       astree := Parser.parse_program Scanner.scan_token lexbuf
107     with
108       _ -> ()
109   ) in
110   let zer = readfile inpt in
111   printf "Syntactic Analysis ...\n";
112   parse zer;
113   (* Uncomment the following lines to perform the contextual analysis and
114      the generation of code *)
115
116   if (ErrorReporter.num_errors() == 0 && !astree != Null_program) then
117   begin
118     printf "Contextual Analysis ...\n";
119     dastree := Checker.check_program !astree;
120     write_file "tabla.xml"
121     (* Uncomment the generation of code *)
122     (*
123     if (ErrorReporter.num_errors() == 0) then
124     begin
125       printf "Code Generation ...\n";
126       Encoder.encode_program !dastree outpt
127     end
128     *)
129   end;
130
```


6.2 Modo de compilación del proyecto

Se utilizó un archivo de configuración con los siguientes comandos:

```
@echo off
```

```
mkdir temp
```

```
copy Code\Misc\ErrorReporter\*. * temp
```

```
copy Code\Misc\Printers\*. * temp
```

```
copy Code\SyntacticAnalyzer\*. * temp
```

```
copy Code\CodeGenerator\*. * temp
```

```
copy Code\ContextualAnalyzer\*. * temp
```

```
copy Code\Compiler.ml temp
```

```
cd temp
```

```
ocamlc -c ErrorReporter.mli
```

```
ocamlc -c RuntimeEntity.mli
```

```
ocamlc -c Ast.mli
```

```
ocamlc -c TreeDrawer.mli
```

```
ocamlc -c Token.mli
```

```
ocamlc -c Parser.mli
```

```
ocamlc -c Scanner.mli
```

```
ocamlc -c Id_entry.mli
```

```
ocamlc -c IdentificationTable.mli
```

```
ocamlc -c Checker.mli
```

```
ocamlc -c TokenPrinter_Pipe.mli
```

```
ocamlc -c TokenPrinter_XML.mli
```

```
ocamlc -c TokenPrinter_HTML.mli
```

```
ocamlc -c IdentificationTablePrinter_XML.mli
```

```
ocamlc -c ErrorReporter.ml
```

```
ocamlc -c TreeDrawer.ml
```

```
ocamlc -c Parser.ml
```

```
ocamlc -c Scanner.ml
```

```
ocamlc -c IdentificationTable.ml
```

```
ocamlc -c Checker.ml
```

```
ocamlc -c TokenPrinter_Pipe.ml
```

```
ocamlc -c TokenPrinter_XML.ml
```

```
ocamlc -c TokenPrinter_HTML.ml
```

```
ocamlc -c IdentificationTablePrinter_XML.ml
```

```
ocamlc -c -pp camlp4o Compiler.ml
```

```
ocamlc ErrorReporter.cmo TreeDrawer.cmo TokenPrinter_Pipe.cmo  
TokenPrinter_XML.cmo TokenPrinter_HTML.cmo Parser.cmo Scanner.cmo  
IdentificationTablePrinter_XML.cmo IdentificationTable.cmo Checker.cmo  
Compiler.cmo -o Triangle.exe
```

```
move Triangle.exe ..  
cd ..  
rd /s /q temp
```