# SymC++: Un depurador simbólico para C++


Proyecto Sistemas Informáticos

**Clara Antolín García**
**Carlos Gabriel Giraldo García**
**Raquel Peces Muñoz**

**Departamento de Sistemas Informáticos y Computación**
**Facultad de Informática**
**Universidad Complutense de Madrid**

**Junio 2014**

Documento maquetado con TEXIS v.1.0+.

Este documento está preparado para ser imprimido a doble cara.

# SymC++: Un depurador simbólico para C++

*Proyecto de Sistemas Infórmaticos*

**II/2014/6**

Clara Antolín García
Carlos Gabriel Giraldo García
Raquel Peces Muñoz
Dirigido por:
Miguel Gómez Zamalloa-Gil

**Departamento de Sistemas Informáticos y Computación**
**Facultad de Informática**
**Universidad Complutense de Madrid**

**Junio 2014**

ISBN

# Autorización

Clara Antolín García, Carlos Gabriel Giraldo García y Raquel Peces Muñoz autores del presente documento y del proyecto "SymC++, una herramienta de depuración simbólica" autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Madrid, a 13 de Junio de 2014

Clara Antolín García Carlos Gabriel Giraldo García Raquel Peces Muñoz

*A nuestros familiares*
*por creer siempre en nosotros*
*y a tí lector,*
*por interesarte en nuestro proyecto*

# Agradecimientos

Este trabajo no sería posible si no hubiésemos contado con todo el apoyo de todas las personas que nos rodean.

En primer lugar agradecer a nuestro tutor Miguel Gómez Zamalloa-Gil, el cual nos ha prestado todo su apoyo, interés y colaboración durante todo el desarrollo a lo largo de estos meses.

En segundo lugar a todos los profesores que nos han acompañado en nuestra formación desde el primer día que pusimos un pie en la facultad, sin ellos no habríamos adquirido los conocimientos que nos han permitido desarrollar este trabajo.

Por último y no menos importante, a nuestros familiares, amigos y compañeros, los cuales nos han soportado y apoyado en los momentos de desánimo y crispación.

A todos ellos, muchas gracias por creer en nosotros.

# Resumen

La ejecución simbólica o evaluación simbólica es un modo de analizar programas para determinar qué entradas causan cada parte del programa a ejecutar. En este tipo de programas, un intérprete recorre el programa donde cada variable asume un valor simbólico acotado. Este recorrido dará como resultado expresiones en términos de dichos símbolos de expresiones y variables y las limitaciones en cuanto a los símbolos de los posibles resultados de cada rama condicional.

Dicho recorrido nos permite determinar las condiciones que deben ser verificadas por los datos de ingreso para que un camino particular se ejecute, y la relación entre los valores ingresados y producidos en la ejecución de un programa.

Durante este proyecto hemos desarrollado una herramienta llamada SymC++ que se basa en la ejecución simbólica para generar test en programas escritos en C++, escribiendo como parámetros el nombre del método, el intervalo de números enteros y la profundidad de los bucles, para ejecutar la función simbólicamente y obtener información de las diferentes ramas generadas. Por otro lado, se podrá obtener información a través de los XMLs generados por el árbol de ejecución o los resultados obtenidos.

El objetivo del proyecto es dotar a los estudiantes de iniciación a la programación de una herramienta que les permita depurar sus programas por un método diferente a la prueba y error.

## Palabras clave

Ejecución simbólica, Evaluación simbólica, C++, XML, depuración, intérprete, valor simbólico.

# Abstract

Symbolic execution or symbolic evaluation is a means of analyzing programs to determine what inputs cause each part of a program to execute. In this kind of programs, an interpreter follows the program where every variable assume a bounded symbolic value. It thus arrives expressions in terms of those symbols for expressions or variables, and constraints in terms of those symbols for the possible outcomes of each conditional branch.

This path allows us to determine the conditions that have to be verified by the input data for a particularly branch is executed, and the relationship between input values and the produced by program execution.

During this project we have developed a tool called SymC++ which is based on symbolic execution to generate test programs written on C++. We have to write like parameters the method name, the range of integers and the deep to bucles, to execute it symbolically and get the information generated from different branches. On the other side, we could get information through the XMLs generated by the execution tree or results.

The project objective is to provide students of programming introduction a tool that allows them to debug their programs by a different method of trial and error.

## Key words

Symbolic execution, Symbolic evaluation, C++, XML, symbolic value

# Índice

**Autorización** v

vii

**Agradecimientos** ix

**Resumen** xi

**Abstract** xiii

**1. Introducción** **1**
1.1. Título del proyecto . . . . . 1
1.2. Motivación de la propuesta . . . . . 1
1.3. Estado del arte . . . . . 2
1.3.1. C++ un lenguaje con historia . . . . . 2
1.3.2. Un mundo hambriento de software . . . . . 2
1.3.3. El software debe estar correcto . . . . . 2
1.4. Objetivos . . . . . 5
1.5. Trabajos relacionados . . . . . 6
1.5.1. PEX . . . . . 6
1.5.2. PET . . . . . 7
1.5.3. jSYX . . . . . 7
1.5.4. jPET . . . . . 7

**2. Herramienta de Clang** **9**
2.1. Introducción . . . . . 9
2.2. AST2XMLtool . . . . . 9
2.3. Historia de LLVM . . . . . 9
2.4. Usando el árbol de sintaxis abstracta . . . . . 10
2.5. Manipulación de un AST con clang . . . . . 11

2.6. Instrucciones como nodos de un fichero XML . . . . . 11
2.7. Bloques de instrucciones . . . . . 11

**3. Intérprete con restricciones 15**
3.1. Introducción . . . . . 15
3.2. Implementacion . . . . . 15
3.3. Prolog . . . . . 16
3.4. Conceptos de Prolog . . . . . 17
3.5. Aspectos relevantes de la implementación . . . . . 17
3.6. Diseño del intérprete . . . . . 17
3.7. Ciclo de ejecución del intérprete . . . . . 19

**4. Herramienta SymC++ 23**
4.1. Introducción . . . . . 23
4.2. Diseño . . . . . 23
4.3. Funcionalidades . . . . . 24
4.4. Conexión con el resto de secciones . . . . . 24
4.5. Directorio de archivos . . . . . 25

**5. Conclusiones y trabajo futuro 27**
5.1. Conclusiones . . . . . 27
5.2. Ampliaciones potenciales y trabajo futuro . . . . . 28

**A. Manual de usuario 31**
A.1. Requisitos previos . . . . . 31

**Bibliografía 33**

# Índice de figuras

2.1. Figura Ejemplo del AST que emplea Clang a partir de un código específico. . . . . 10
2.2. Figura con ejemplo de la disposición básica de un nodo function. 11
2.3. Figura utilizada para marcar una imagen por hacer. . . . . 12
2.4. Figura utilizada para marcar una imagen por hacer. . . . . 13
2.5. Figura utilizada para marcar una imagen por hacer. . . . . 13
2.6. Figura utilizada para marcar una imagen por hacer. . . . . 14
2.7. Figura utilizada para marcar una imagen por hacer. . . . . 14

3.1. Figura con ejemplo de archivo XML de entrada. . . . . 18
3.2. Figura utilizada para marcar una imagen por hacer. . . . . 19
3.3. Figura utilizada para marcar una imagen por hacer. . . . . 19
3.4. Figura utilizada para marcar una imagen por hacer. . . . . 20

# Índice de Tablas

# Capítulo 1

# Introducción

## 1.1. Título del proyecto

Desarrollo de una herramienta de depuración simbólica para las asignaturas de iniciación a la programación en C++ en las facultades de Informática y Estudios Estadísticos.

## 1.2. Motivación de la propuesta

El proyecto nace de la idea y la necesidad de utilizar una herramienta útil para los estudiantes de iniciación a la programación, en este caso concreto en el lenguaje C++ por ser el lenguaje que se utiliza en los primeros curos de los grados de las Facultades de Informática, Estudios Estadísticos y Matemáticas.

El problema al que se enfrentan los estudiantes es cuando los problemas a resolver crecen en complejidad, los programas también y normalmente no se comportan como deben para todos los casos. Escribir programas totalmente correctos requiere el uso de una serie de metodologías que en general resultan muy complejas para los estudiantes en los primeros cursos. La mayoría de ellos siguen una mecánica de prueba-error, es decir, escriben los programas sin pararse a pensar demasiado acerca de su corrección, y luego los prueban para ver si se comportan como esperan. Estas pruebas normalmente no son lo suficientemente exhaustivas y por lo tanto los estudiantes normalmente no son ni siquiera conscientes de las incorrecciones de sus programas. Desgraciadamente, los mecanismos y herramientas que ayudan en el testing y depuración de programas son en general demasiado avanzados para poder ser utilizados por programadores inexpertos.

Dado este problema aparece nuestro proyecto, como una herramienta sencilla e intuitiva que pueda mostrar de un modo sencillo el recorrido de un programa por sus diferentes ramas y los resultados que obtendría en cada uno de ellos.

## 1.3. Estado del arte

### 1.3.1. C++ un lenguaje con historia

C++ es un lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, el C++ es un lenguaje híbrido.Posteriormente se añadieron facilidades de programación genérica, que se sumó a los otros dos paradigmas que ya estaban admitidos (programación estructurada y la programación orientada a objetos). Por esto se suele decir que el C++ es un lenguaje de programación multiparadigma. Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes, tales como ROOT.

### 1.3.2. Un mundo hambriento de software

Actualmente el software es uno de los productos más requeridos en el mundo. Tal demanda de software implica procesos de desarrollo más intensos, exhaustivos y menos prestos a errores. Siendo la corrección de fallos la etapa en la que más recursos se invierten y en la que hay más interés por mejorar su eficiencia.

### 1.3.3. El software debe estar correcto

A día de hoy en el mercado existen paquetes de herramientas capaces de ofrecer a los desarrolladores detección de errores y estudios sobre la ejecución de sus programas. Existen varias maneras de verificar la correctitud de un programa:

#### 1.3.3.1. Revisiones (Software Reviews)

Hechas por personas, por lo general involucradas en el proceso de desarrollo, expertas en la materia. Son reuniones en las que los revisores analizan el código y lo estudian en grupo con el objetivo de encontrar errores. Existen protocolos estandarizados por la asociación IEEE que rigen el orden de estas revisiones. Dependiendo de los conductores del análisis y del objetivo de este, cambia la denominación de la revisión.

##### Peer Review

Realizada por el propio autor del código o un compañero y su objetivo es evaluar el contenido técnico y la calidad del software.

##### Management Reviews

Cuentan con la asistencia de los directivos o superiores, el objetivo es comprobar el trabajo realizado y sopesar el trabajo futuro.

##### Audit Reviews

La realiza personal externo al proyecto y busca irregularidades en cuanto a las especificaciones, estándares o acuerdos empresariales.

#### 1.3.3.2. Análisis dinámico

Supone la ejecución del programa y observar su comportamiento. Para que el anáilisis sea efectivo de debe realizar con los suficientes casos de prueba. Estos casos de prueba representan los comportamientos que se esperan del programa en distintas condiciones de ejecución. Para asegurar la efectividad de las pruebas se pueden usar distintas técnicas como estudiar la cobertura del código (comprobar que en diferentes ejecuciones de un programa se pasa por todas las instruccciones al menos una vez) o el uso de programas fuzzers que ayudan a asegurar que una porción adecuada del conjunto de posibles comportamientos del programa ha sido observada.

#### 1.3.3.3. Análisis estático

Consiste en el análisis, generalmente automático, por parte de una herramienta sobre el programa sin que este se ejecute. Puede realizarse sobre el código fuente o sobre el código objeto. La industria ha reconocido los métodos de análisis estático como elementos clave a la hora de mejorar la calidad

de programas complejos, siendo usos muy generalizados en el ámbito de la medicina, la aviación y la energía nuclear, campos en los que no se pueden permitir errores ni riesgos. En la mayoría de los casos se emplea con el objetivo de localizar errores de codificación o vulnerabilidades en el código, pero también se incluyen una variedad de métodos formales que verifican ciertas propiedades del programa.

### Verificacion de modelos(Model cheking)

Método en el cual el programa y su especificación se expresan matemáticamente con el objetivo de verivicar su correctitud.

### Lógica de Hoare

Se emplea un sistema con un conjunto de reglas lógicas con las que razonar rigurosamente la correctitud del programa.

### Análisis del flujo de datos

Es una técnica en la que se recoge información acerca del conjunto de posibles valores calculados en distintos puntos del programa. Se utiliza para estudiar a que partes del programa se podría propagar un valor en particular asginado a una variable.

### Interpretación abstracta

Es un método que modela el efecto de las instrucciones del programa en la máquina mediante una "ejecución" basada en las propiedades matemáticas de cada instrucción. La máquina sobre la que ésta se lleva a cabo es una representación simplificada de la máquina verdadera. Esta máquina abstracta aproxima el comportamiento del sistema haciéndolo más fácil de analizar.

### Ejecución simbólica

Es un caso particular de Interpretación abstracta. Consiste en analizar el programa para luego concluir qué conjunto de entradas ejecutan qué partes del código. Un intérprete recorre el programa asumiendo valores simbólicos para las entradas en vez de solicitarlas al usuario, llegando al final con un conjunto de expresiones en términos de esos valores simbólicos de los cuales deduce los posibles valores reales. Este procedimiento aplicado al hardware recibe el nombre de Simulación simbólica.

## 1.4. Objetivos

En este proyecto se estudiará y desarrollará una herramienta de "depuración simbólica" que permitirá estudiar el funcionamiento de programas informáticos sin ser ejecutados, a base de observar todas sus posibles ramas de ejecución (hasta un cierto nivel) así como los correspondientes pares entrada-salida. Dicha herramienta podría ayudar enormemente a los estudiantes de iniciación a la programación, y en general a programadores inexpertos, a la hora de razonar acerca de la corrección de sus programas.

Sólo se utilizará como forma de comprobación, no como solucionador de errores ni sintácticos ni semánticos, es decir, no realiza la función de compilador; mostrará los resultados de ciertas posibles entradas para comprobar si ésa es la solución que esperabas, de no ser así, el alumno sabrá que el código estará mal planteado. Los alumnos que están empezando a programar en C++ saben si su programa compila gracias al entorno que estén utilizando pero les es más difícil de comprobar si su programa está correcto sin la ayuda de una interfaz o de algún medio para eso. Ahí es donde entra nuestra herramienta.

La herramienta contará con una interfaz de usuario, proporcionando varias ventajas adicionales: el alumno podrá ver y actualizar cómodamente, en todo momento, el código que desee probar, podrá tanto escribir un código desde cero como importar un código ya creado, por otro lado, la herramienta marcará el recorrido que se ha realizado de forma que el alumno podrá corregir posibles errores más rápidamente, por ejemplo, al encontrar un recorrido que no era el deseado.

Muchas veces, para ayudarnos en la búsqueda de posibles fallos, es aconsejable programar en módulos, es decir, utilizando subprogramas para encapsular el código mejor. Con nuestra herramienta podrás elegir el nombre de la función que quieres probar aunque haya muchas funciones escritas, esto es útil ya que podrás ir comprobando sólo las que quieras sin tener que cambiar el código.

Este proyecto es muy ambicioso y posee una gran envergadura que supera las posibilidades de un proyecto de fin de carrera para ser abordado desde cero. Es por esto que nosotros, durante el desarrollo de este proyecto planteamos una herramienta que debe ser continuada en cursos posteriores para poder acaparar todas la sintaxis de C++ y para que pueda ser considerada una herramienta de depuración completa.

## 1.5. Trabajos relacionados

Como hemos mencionado anteriormente la ejecución simbólica, no es algo nuevo y ha adquirido protagonismo en los últimos años. Es por esto que podemos encontrar varios proyectos orientados al testing de diversos estilos, aquí describimos algunos de ellos para contextualizar nuestro proyecto.

### 1.5.1. PEX

PEX es una herramienta desarrollada por Microsoft par a la generación automática de casos de prueba. Esta herramienta genera entradas de test para programas .NET, por tanto puede analizar cualquier programa que se ejecute en una máquina virtual .NET y soporta lenguajes como C#, Visual Basic y F#.

El enfoque de PEX se caracteriza por implementar la ejecución dinámica simbólica, pero también permite el uso de ejecucución concreta de valores para simplificar las restricciones, las cuales serían resueltas por el resolutor SMT. Por tanto, mediante un proceso iterativo, PEX realiza una ejecución concreta del método a analizar y examina la traza de ejecución buscando ramas no exploradas. Una vez ha encontrado una rama no explorada, usa la ejecu-ción simbólica y un sistema resolutor de restricciones para generar valores concretos que exploren dicha rama. Este proceso se repite hasta obtener el recubrimiento deseado.

El hecho de combinar la ejecución simbólica con la concreta permite en muchos casos incrementar la escalabilidad y tratar con situaciones donde la ejecución no depende sólo del código sino de factores externos. Se entiende por factores externos el uso de librerías nativas, llamadas al sistema operativo o interacciones con el usuario. Ante este tipo de situaciones, la ejecución simbólica presenta grandes limitaciones y en general, no se puede aplicar.

Esta herramienta se puede integrar en el entorno de desarrollo de Visual Studio facilitando su uso como un añadido (Addon).

Como proyecto interesante relacionado con PEX existe el juego Code Hunt también desarrollado por Microsoft. En este juego el jugador debe escribir código para avanzar en el juego. La relacióncon PEX es que lo que se muestra al jugador son las entradas y salidas, y mediante ejecución simbólica se evaluará el código escrito y de este modo poder avanzar.

### 1.5.2. PET

PET (Partial Evaluation-based Test Case Generator for Bytecode) es una herramienta cuyo propósito es generar casos de prueba de forma automática para programas escritos en bytecode (código de bytes de Java). PET adopta el enfoque previamente comentado, esto es, ejecuta el bytecode simbólicamente y devuelve como salida un conjunto de casos de prueba (test-cases). Cada caso de prueba está asociado a una rama del árbol y se expresa como un conjunto de restricciones sobre los valores de entrada y una descripción de los contenidos de la memoria dinámica (o heap).

Las restricciones de la memoria dinámica imponen condiciones sobre la forma y contenidos de las estructuras de datos del programa alojadas en esta misma. PET utiliza de un resolutor de restricciones que genera valores concretos a partir de estas restricciones, permitiendo la construcción de los tests propiamente dichos.

PET puede usarse a través de una interfaz de línea de comandos o bien usando una interfaz web. Además soporta una variedad de opciones interesantes, como la elección de criterios de recubrimiento o la generación de tests jUnit. Estas opciones se describen con más detalle en el segundo capítulo.

### 1.5.3. jSYX

jSYX es un proyecto desarrollado el curso pasado como trabajo de Sistemas Informáticos. Este proyecto se basa en una máquina virtual de Java que permite la ejecución simbólica de archivos .class de Java, de este modo puede ser utilizado para el desarrollo automático de Tests.

El enfoque de jSYX es el uso de del Bytecode de Java como lenguaje de ejecución. Esta herramienta permite la ejecución recibiendo como parámetros una clase y su método, y de este modo permite la ejecución simbólicamente o de manera concreta. Por otra lado se podrá obtener información sobre el bytecode del archivo .class.

### 1.5.4. jPET

Al igual que el detallado en la sección anterior, jPET también se trata de un proyecto de Sistemas Informáticos, en este caso desarrollado durante el curso 2010/2011. Este proyecto se basa en PET, la herramienta detallada anteriormente, y es una extensión de dicha herramienta para poder utilizarse en programas Java de alto nivel e integrarse en Eclipse, con el objetivo de poder usar los resultados obtenidos por PET durante el proceso de desarrollo de software.

El enfoque de jPET es el tratamiento de la información generada por PET con el objetivo de presentarla al usuario de una manera más fácil, sencilla e intuitiva. Es por esto que incorpora un visor de casos de prueba para mostrar el contenido de memoria antes y después de la ejecución, una herramienta de depuración, para ver los resultados mostrando la secuencia de instrucciones que el caso de prueba ejecutaría, y es capaz de analizar sintácticamente precondiciones de métodos para evitar la generación de casos de prueba poco interesantes.

# Capítulo 2

# Herramienta de Clang

## 2.1. Introducción

La herramienta ast2xml conforma junto con el xmlinterpreter el cuerpo central del proyecto. Si bien ast2xml no cumple el papel más importante, suple las necesidades básicas de obtener a partir del código fuente una representación más manejable y sencilla de visualizar. Ha sido desarrollada a partir del proyecto abierto e internacional LLVM, pero se apoya principalmente en su compilador Clang.

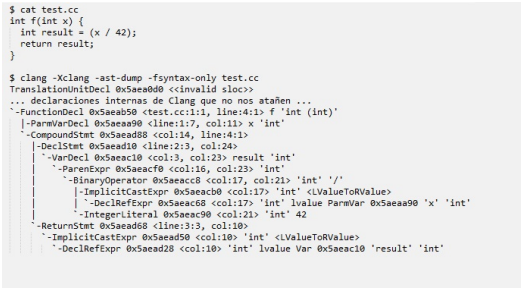
## 2.2. AST2XMLtool

Al inicio del curso nos dimos cuenta de que nos podíamos aproximar al problema de distintas formas. Si bien, de una forma similar al proyecto Jsyx teniamos la posibilidad de basar nuestro proyecto en el uso de algo similar al bytecode de java pero relacionado con C++.

## 2.3. Historia de LLVM

LLVM nació como proyecto en el año 2000 en la Universidad de Illinois en Urbana-Champaign. En 2005, junto con Apple Inc., se empezó a adaptar el sistema para varios usos dentro del ecosistema de desarrollo de Apple. Actualmente LLVM está integrado en las últimas herramientas de desarrollo de Apple para sus sistemas operativos.

El un principio "LLVM" eran las iniciales de "Low Level Virtual Machine" (Máquina Virtual de Bajo Nivel), pero esta denominación causó una confu-



```
$ cat test.cc
int f(int x) {
  int result = (x / 42);
  return result;
}

$ clang -Xclang -ast-dump -fsyntax-only test.cc
TranslationUnitDecl 0x5aea0d0 <<invalid sloc>>
... declaraciones internas de Clang que no nos atañen ...
`-FunctionDecl 0x5aeab50 <test.cc:1:1, line:4:1> f 'int (int)'
  |-ParmVarDecl 0x5aeaa90 <line:1:7, col:11> x 'int'
  `-CompoundStmt 0x5aead88 <col:14, line:4:1>
    |-DeclStmt 0x5aead10 <line:2:3, col:24>
    | `-VarDecl 0x5aeac10 <col:3, col:23> result 'int'
    |   `-ParenExpr 0x5aeacf0 <col:16, col:23> 'int'
    |     `-BinaryOperator 0x5aeacc8 <col:17, col:21> 'int' '/'
    |       |-ImplicitCastExpr 0x5aeacb0 <col:17> 'int' <LValueToRValue>
    |       | `-DeclRefExpr 0x5aeac68 <col:17> 'int' lvalue ParmVar 0x5aeaa90 'x' 'int'
    |       `-IntegerLiteral 0x5aeac90 <col:21> 'int' 42
    `-ReturnStmt 0x5aead68 <line:3:3, col:10>
      `-ImplicitCastExpr 0x5aead50 <col:10> 'int' <LValueToRValue>
        `-DeclRefExpr 0x5aead28 <col:10> 'int' lvalue Var 0x5aeac10 'result' 'int'
```

Figura 2.1: Figura Ejemplo del AST que emplea Clang a partir de un código específico.

sión ampliamente difundida, puesto que las máquinas virtuales son solo una de las aplicaciones de LLVM. Cuando la extensión del proyecto se amplió incluso más, LLVM se convirtió en un proyecto que engloba una gran varriedad de otros compiladores y tecnologías de bajo nivel. Por tanto, el proyecto abandonó las iniciales y actualmente, LLVM es la manera de referirse a todas esas utilidades.

El amplio interés que ha recibido LLVM ha llevado a una serie de tentativas para desarrollar front-ends totalmente nuevos para una variedad de lenguajes. El que ha recibido la mayor atención es Clang, un nuevo compilador que soporta otros lenguajes de la familia de C (Objective-C, C++, etc..). Apoyado principalmente por Apple, se espera que Clang sustituya al comiplador del sistema GCC.

## 2.4. Usando el árbol de sintaxis abstracta

La herramienta AST2XML efectúa como primer paso en la adaptación del código su arbol de sintaxis abstracta (Abstract Syntax Tree).

Un árbol de sintaxis abstracta (AST) es una representación de árbol de la estructura sintáctica abstracta (simplificada) del código fuente ampliamente utilizada en los distintos compiladores del mercado. Cada nodo del árbol denota una construcción que ocurre en el código fuente. La representación del AST que emplea Clang se diferencia de la de otros compiladores en cuanto a que mantiene la semejanza con el código escrito y con el estándar de C++.

Esta estructura es obtenida mediante el uso de las distintas utilidades incluidas en el apartado de desarrollo de Clang. Mediante dichas facilidades se recorre el AST de forma recursiva conservando los detalles más importantes del código en un fichero que se devuelve al usuario.

```
<function name="foo" type="int" line="3">
    <params>
        <param type="int" name="a"/>
        <param type="int" name="b"/>

    </params>
    <body>
        ......
    </body>
</function>
```

Figura 2.2: Figura con ejemplo de la disposición básica de un nodo function.

## 2.5. Manipulación de un AST con clang

Como se puede apreciar en la figura, la representación que se obtiene directamente de clang puede llegar a parecer abrumadoramente compleja y redundante. En momentos iniciales del proyecto tomamos la decisión de que información como la ubicación de los elementos, los tipos de las variables, o incluso el tipo de los nodos no serían necesarias en cada momento. Decidimos pues optar por una representación más acotada, clara y concisa que, expresando lo mismo, fuera lo suficientemente sencilla de manipular por el intérprete de prolog.

## 2.6. Instrucciones como nodos de un fichero XML

El fichero que se le devuelve al usuario tiene formato XML y contiene información acerca de las funciones que, en el código fuente, haya declarado el usuario. Dichas funciones se encuentran englobadas en nodos function que almacenan el nombre, el número de línea, el tipo devuelto, los parámetros y el cuerpo de la función en cuestión. De los parámetros registramos su tipo y el nombre de su declaración.

En el desarrollo del proyecto hemos tenido que, de la misma forma, asignar a cada declaración, instrucción y expresión de C++ contempladas en nuestro dominio un nodo XML que mantuviera su significado y de la misma forma no incrementara la complejidad del texto.

## 2.7. Bloques de instrucciones

Una noción que existe en el AST de Clang es la del CompoundStatement que representa a un bloque de código delimitado por llaves (..). Puesto que


Figura 2.3: Figura utilizada para marcar una imagen por hacer.

esperamos que el usuario haga uso de ellos sólo en compañia de otras estructuras (if, while, for o funciones), en el XML se representa con el nodo body. Que contiene los nodos de las instrucciones programadas.

## Declaraciones e inicializaciones

C + + es un lenguaje fuertemente tipado, y requiere que cada variable esté declarada junto con su tipo antes de su primer uso. En un caso práctico, esto informa al compilador el tamaño reservar en memoria para la variable y la forma de interpretar su valor. Si se declaran varias variables del mismo tipo, se puede realizar en una sola instrucción.

Cuando se declara una variable, adquiere un valor indeterminado hasta que se le asigne alguno explícitamente. En C++ se contemplan tres maneras de inicializar el valor de una variable, aunque en nuestro proyecto, de cara a el uso de nuestra herramienta por parte de alumnos que están aprendiendo a programar, admitimos únicamente la forma más sencilla.

Los nodos declaration contienen información acerca del tipo de la variable, su nombre, la línea en la que se encuentra la declaración y la expresión cuyo valor se le va a asignar.

## Estructuras de control

Las instrucciones que producen ramificaciones en la ejecución de un programa, tales como: el if, el while o el for. Para poder realizar su ejecución correctamente mantenemos en sus respectivos nodos sus señas más características.


Figura 2.4: Figura utilizada para marcar una imagen por hacer.


Figura 2.5: Figura utilizada para marcar una imagen por hacer.

El nodo de la instrucción if contiene el número de línea, y los nodos correspondientes a la condición que dirige su ejecución, al cuerpo del then y al cuerpo del else (en caso de estar especificado en el código).

La instrucción while se traduce a un nodo que alberga como atributo el número de línea donde comienza la instrucción, y los nodos de la expresión de su condición y el bloque de instrucciones de su cuerpo.

La estructura del nodo de la instrucción for mantiene el número de línea, y en su interior los nodos de la declaración de su variable de control, la condición, la instrucción de avance y el cuerpo del bucle.

## Expresiones

Las expresiones escritas en C++ se traducen a nuestro formato en XML en nodos descritos por sus operadores, la variable a la que hacen referencia o la constante que representan.


Figura 2.6: Figura utilizada para marcar una imagen por hacer.


Figura 2.7: Figura utilizada para marcar una imagen por hacer.

En caso de que estos nodos hagan referencia a un operador, contienen información acerca del tipo de este (binario, unario, asignación, etc...) y de las expresiones que contienen.

Las expresiones que se refieren a variables almacenan el nombre de estas y de forma equivalente las que representan constantes guardan su valor.

Un caso particular de las expresiones sería el de las llamadas a función. Dado que en el subconjunto de C++ en el que se mueve nuestra herramienta tan sólo contemplamos la utilización de funciones, la llamada a otros métodos.

## Operadores de asignación

Para representar las instrucciones de asignación nos expresamos mediante los nodos assignment y assignmentOperator, conteniendo el nombre de la variable, la línea y la expresión de la cual se calcula el valor.

# Capítulo 3

# Intérprete con restricciones

## 3.1. Introducción

En este capítulo hablaremos sobre la parte principal de nuestro proyecto. Esta consiste en el intérprete que recibe el árbol sintáctico de la herramienta AST2XML. El cometido del intérprete, a grandes rasgos, será el de realizar una ejecución simbólica del código devolviendo un conjunto de valores de entrada y salida que se corresponden con lo programado..

## 3.2. Implementacion

Como ya hemos indicado antes, AST2XML devuelve un archivo en formato XML con una versión simplificada del árbol sintáctico que emplea Clang con el que realizaremos la ejecución simbólica. Para llevar a cabo una ejecución simbólica es necesario tener un sistema capaz de establecer restricciones lógicas y matemáticas y que aparte las pueda resolver.

## Programación por restricciones

La Programación por restricciones es un paradigma de la programación en informática donde las las variables está relacionadas mediante restricciones (ecuaciones). Se emplea en la descripción y resolución de problemas combinatorios, especialmente en las áreas de planificación y programación de tareas (calendarización). La programación con restricciones se basa principalmente en buscar un estado en el cual una gran cantidad de restricciones sean satisfechas simultáneamente, expresándose un problema como un conjunto de restricciones iniciales a partir de las cuales el sistema construye las relaciones

que expresan una solución. Actualmente existen muchos frentes de desarrollo relacionados con la programción con restricciones. Entre ellos destacan: Oz: Lenguaje multiparadigma y esotérico basado en la rama concurrente de la programación por restricciones. En él que se expresa música a partir de unas restricciones expresadas explícitamente por el programador. Se utiliza en proyectos tales como Mozart y Strasheela. Choco: Es una librería que añade satisfacción de restricciones a Java. Está construida en una estructura basada en la propagación de eventos. Ha sido utilizada en otros proyectos de ejecución simbólica como JsyX. Gecode: Es un proyecto abierto que cuenta con un conjunto de herramientas basado en C/C++ para el desarrollo de sistema y aplicaciones nativas que se apoyen en restricciones.

http://ktiml.mff.cuni.cz/ bartak/downloads/WDS99.pdf La programación por restricciones está intimamente relacionada con la programación lógica. Por ende existe la programación lógica con restricciones. Ésta es una extensión de la programación lógica en la que se añade el concepto de satisfacción de restricciones.

Para el desarrollo de nuestro intérprete optamos por el lenguaje de programación lógica Prolog. En gran parte debido a la familiaridad que nos supone y al conjunto de facilidades relacionadas con la programación por restricciones que ya conocíamos de antemano

## 3.3. Prolog

La Programación Lógica tiene sus orígenes más cercanos en los trabajos de prueba automática de teoremas de los años sesenta. J. A. Robinson propone en 1965 una regla de inferencia a la que llama resolución, mediante la cual la demostración de un teorema puede ser llevada a cabo de manera automática. La resolución es una regla que se aplica sobre cierto tipo de fórmulas del Cálculo de Predicados de Primer Orden, llamadas cláusulas y la demostración de teoremas bajo esta regla de inferencia se lleva a cabo por reducción al absurdo. La realización del paradigma de la programación lógica es el lenguaje Prolog

Prolog es un lenguaje de programación ideado a principios de los años 70 en la Universidad de Aix-Marseille I (Marsella, Francia). No tiene como objetivo la traducción de un lenguaje de programación, sino la clasificación algorítmica de lenguajes naturales. [1]

## 3.4. Conceptos de Prolog

La programación lógica tiene sus raíces en el cálculo de predicados. Es un conjunto de cláusulas de la forma: "q :- p", es decir, si p es cierto entonces q es cierto. Una cláusula pueden ser un conjunto de hechos positivos, por ejemplo de la forma "q :- p, r, s." ; una implicación con un único consecuente, "q :- p" ; un hecho positivo, "p." ; instrucciones con parámetros de Entrada/Salida "p(Entrada,Salida)." ; o incluso llamadas a otras funciones.

En Prolog, los predicados se contrastan en orden , la ejecución se basa en dos conceptos: la unificación y el backtracking. Una vez ejecutamos una función de Prolog se sigue ejecutando el programa gracias a las llamadas a predicados, si procede, hasta determinar si el objetivo es verdadero o falso. Todos los objetivos terminan su ejecución en éxito ("verdadero"), o en fracaso ("falso"). Si el resultado es falso entra en juego el backtracking, es decir, deshace todo lo ejecutado situando el programa en el mismo estado en el que estaba justo antes de llegar al punto de elección, ahí se toma el siguiente punto de elección que estaba pendiente y se repite de nuevo el proceso, de ahí la utilidad de prolog en nuestro proyecto.

## 3.5. Aspectos relevantes de la implementación

Hemos utilizado la programación lógica basada en restricciones que consiste en reemplazar la unificación de términos de la programación lógica por el manejo de restricciones en un dominio concreto, es un algoritmo especializado e incremental ya que cuando se añade una nueva restricción a un conjunto de restricciones ya resuelto, el resolvedor de restricciones no tiene que resolver el conjunto de restricciones S desde el principio.

Una ventaja de la incorporación de las restricciones al lenguaje de programación es la sencillez al mostrar restricciones como datos de salida. La programación lógica basada en restricciones sobre dominios finitos (use_module(library(clpfd))) proporciona una aritmética completa para variables restringidas a valores enteros o elementos atómicos.

## 3.6. Diseño del intérprete

El intérprete está encapsulado en dos módulos para hacer más fácil su diseño: VariablesTable.pl e Interpreter.pl.

```
<function name="llamadaFactorial" type="int" line="79">
    <params>
        <param type="int" name="a"/>
    </params>
    <body>
        <return line="80">
            <callFunction name="factorial" type="int">
                <arg>
                    <variable name="a"/>
                </arg>
            </callFunction>
        </return>
    </body>
</function>
```

Figura 3.1: Figura con ejemplo de archivo XML de entrada.

## VariablesTable.pl

El intérprete irá guardando las variables y su respectivo valor en una tabla de variables representado en prolog mediante una lista, de forma que todas las posibles operaciones que se puedan aplicar sobre ella estén encapsuladas en un mismo módulo. Por ejemplo, funciones como añadir un elemento a la tabla (add), obtener el valor de una variable (getValue), modificar el nombre de una variable (updateNames), etc. son funciones que sólo se aplican sobre la tabla de variables.

## Interpreter.pl

Por otro lado, este módulo representa el Intérprete que será llamado por la interfaz y el que devolverá la solución. Los parámetros de entrada son: fichero de entrada, fichero de salida, Inf, Sup, MaxDepth y nombre de función.

Fichero de entrada Indica el nombre del fichero .xml donde está el código traducido por clang.

Fichero de salida Indica el nombre del fichero .xml de salida que el intérprete devolverá con las soluciones para una determinada función.

Inf, Sup y MaxLoop Inf y Sup son los valores que representan el límite del dominio de los valores de entrada de la función a interpretar. MaxLoop representa el valor del número máximo de veces que puede ejecutarse una estructura de control: while y for.

Nombre de función Para indicar el nombre de la función que vamos a querer probar entre todas las funciones posibles contenidas en el fichero pasado en “Fichero de entrada”.


Figura 3.2: Figura utilizada para marcar una imagen por hacer.


Figura 3.3: Figura utilizada para marcar una imagen por hacer.

## 3.7. Ciclo de ejecución del intérprete

El primer paso en la ejecución es la lectura del fichero xml y la conversión de la estructura en nodos a una de listas mediante la función que nos aporta la librería sgml:

nos saldría una lista así:

Teniendo esto en cuenta simplificamos . Cada element tiene tres argumentos: el nombre del nodo, los atributos del nodo y el cuerpo del nodo. De esta forma buscaremos previamente la función que el usuario nos haya indicado e iremos pasando por cada una de las instrucciones del código de dicha función y ejecutándolas mediante las sucesivas llamadas a la función execute.


Figura 3.4: Figura utilizada para marcar una imagen por hacer.

## execute/3

Es un predicado cuyos argumentos son: el estado previo a la ejecución de unas instrucciones, la lista que contiene las instrucciones y el estado posterior a la ejecución. Este predicado se planteó de forma que pudiéramos controlar cada una de las posibles instrucciones que puede llegar a haber en el código. Para conseguirlo en cada una de estas posibilidades expresamos explícitamente los casos de forma que la instrucción que se ejecuta en un momento dado es el primer elemento de la lista de instrucciones que recibe la función. Una vez se hayan completado los pasos que simulan el comportamiento de dicha instrucción se hace una "llamada" recursiva a execute con el estado resultante de la ejecución, el resto de la lista de instrucciones y el estado posterior.

Esta forma de realizar los pasos de cómputo se asemeja, en un aspecto más teórico, a la semántica de paso corto (semántica operacional), aunque dicha similitud no es completa puesto que al trabajar con instrucciones puntuales llevamos a cabo muchos pasos intermedios.

Es importante destacar que el bactracking inherente de prolog se restringe en el predicado execute. Esto se debe a que el orden de las instrucciones es siempre el mismo. En el intérprete hemos limitado el backtracking a la evaluación de condiciones puesto que éstas son las que realmente definen el flujo del programa. Es por ello que en las instrucciones de control llegamos a necesitar la presencia de otros predicados que además de dirigir el comportamiento de la ejecución permiten limitar el número de repeticiones que realiza un bucle.

## executeLoop/5

## executeBranch/4

## resolveExpression/4

## work/3

# Capítulo 4

# Herramienta SymC++

...

...

## 4.1. Introducción

En este capítulo explicaremos brevemente cual es la arquitectura de la interfaz desarrollada y su relación como núcleo central y enlace con todos los componentes de la herramienta. Este capítulo está dirigido a aquellas personas interesadas en conocer las decisiones de diseño que hemos tomado, así como la estructura y funcionalidades implementadas.

Este capítulo no pretende ser un manual de usuario o de instalación, el cual se encuentra en la sección de apéndices.

## 4.2. Diseño

A la hora de desarrollar la interfaz para el cliente, el principal problema al que nos enfrentamos era la búsqueda de una interfaz intuitiva, fácil y que ofreciera todas las funcionalidades de depuración que buscamos para los estudiantes a los que está orientada esta herramienta. Es por esto que buscamos una vista de cliente, en la que toda la información resultante de la ejecución simbólica esté disponible en la vista principal. Para el desarrollo de la interfaz hemos utilizado una librería de Synthetica Look And Feel, librería bajo la licencia bajo licencia gratuita para usos no comerciales. Esta librería nos ha permitido dar una mayor vistosidad a la aplicación además de facilitarnos algunas cuestiones relativas al diseño. Por otro lado, todas las

imágenes referentes a los botones también están bajo licencia gratuita para usos no comerciales.

Para la estructura de la interfaz, hemos utilizado un MigLayout, el cual se encuentra bajo licencia pública GNU GPL. Este layout nos facilita la distribución de la pantalla y del mismo modo, ofrece una mayor simplicidad y escalabilidad para la herramienta.

## 4.3. Funcionalidades

Las funcionalidades referentes a la interfaz del cliente son las básicas de cualquier editor de texto e incluso compilador de código. Hay que partir de la base, de que en nuestro proyecto lo más relevante no es la interfaz, sino todo el trabajo que hay detrás con la ejecución simbólica.

Como funcionalidades de depuración cabe destacar la visión en formato árbol de los XMLs generados tanto por Clang como por Prolog, los cuales contienen toda la información relativa al árbol sintáctico anotado del archivo C++ y al resultado de la ejecución simbólica respectivamente.

Referente a la visión del área de texto, para darle un aspecto de editor de código hemos añadido la visibilidad del número de línea.

Por tanto, para implementar las funcionalidades descritas, así como otras varias hemos utilizado las siguientes clases de código, todas bajo la licencia GNU: LineNumbers.java [1] (http://javaknowledge.info/jtextpane-line-number/) MultiLineCellRenderer.java[2] (http://www.java2s.com/Code/Java/Swing-Components/MultiLineCellExample.htm) XML2JTree.java [3] (cse.unl.edu/ reich/XML/DOM/XML2JTree.java)

## 4.4. Conexión con el resto de secciones

Como ya se ha indicado, la interfaz sirve de enlace entre la herramienta Clang, el intérprete Prolog y el usuario. Puesto que las llamadas a estas partes sólo se realizan puntualmente y de un único modo, es decir, cuando el usuario decide lanzar la ejecución simbólica, se han creado scripts con comandos shell, los cuales son llamados desde java como procesos externos.

Hay que tener en cuenta, que la herramienta pedirá al inicio que se seleccione dónde se encuentra instalada la herramienta Clang, la cual se guardará en un archivo de configuración. Por tanto, al lanzar la ejecución, tomará dicho directorio como herramienta y será la encargada de generar el archivo XML del árbol de ejecución.

## 4.5. Directorio de archivos

El sistema de archivos de la aplicación está dividido en varios directorios: files : Directorio en el que se almacenan todos los archivos generados, tanto los XMLs generados por Clang y el intérprete, como el archivo de c++ que se va a ejecutar. En este directorio, se incluyen los archivos "BuiltinsIO.h" y "BuiltinsSTD.h" necesarios para el uso de las funciones de entrada y salida. tools: En este directorio se incluyen los archivos prolog necesarios para que el intérprete funcione y los comandos shell que realizan la llamada tanto a la herramienta Clang como al intérprete Prolog. img: Directorio en el que se incluyen todas las imágenes necesarias para la interfaz. libs: Directorio en el que se incluyen todas las librerías necesarias para que la interfaz funcione, las cuales son las referentes al estilo, synthetica, y al layout, miglayout.

# Capítulo 5

# Conclusiones y trabajo futuro

...

...

## 5.1. Conclusiones

### Conclusiones sobre el trabajo realizado

Tras un curso desarrollando este proyecto podemos decir que hemos cumplido los objetivos fijados al inicio del curso, desarrollando una aplicación que realiza la ejecución simbólica de un conjunto limitado de operadores de C++. La herramienta evalúa todas las posibles ramas de funciones sencillas teniendo en cuenta todas las limitaciones de la ejecución.

Los resultados obtenidos por dicha ejecución se muestran de una forma sencilla y visual al usuario, además de mostrar las líneas que recorre dicho intérprete.

### Conclusiones personales

Este proyecto nos ha aportado ciertas capacidades que no hemos podido desarrollar en otras asignaturas. En primer lugar, nos ha permitido enfrentarnos a un proyecto de investigación informática, mucho más teórico que práctico. Donde el mayor trabajo se ha centrado en la investigación de tecnologías que en el desarrollo de las mismas.

Por otro lado, el hecho de enfrentarnos nosotros mismos a un proyecto, a pesar de la tutorización, nos ha obligado a trabajar en equipo, coordinarnos

y organizarnos para cumplir unos objetivos y plazos estipulados por nosotros mismos. Eso nos ha aportado responsabilidad y compromiso para con el proyecto.

A pesar que durante el presente curso hemos cursado otras asignaturas y no hemos tenido la posibilidad de dedicarle más tiempo al proyecto, consideramos que el trabajo realizado ha sido muy productivo y nos ha aportado una gran experiencia académica y personal.

## 5.2. Ampliaciones potenciales y trabajo futuro

Nuestro proyecto puede ser utilizado como base para siguientes cursos para la asignatura de Sistemas Informáticos o proyecto de final de grado. Desde el primer día, nuestro tutor nos puntualizó que este es trabajo para toda una vida, y tener en cuenta toda la sintaxis de C++ es inviable en un curso académico.

### Mayor repertorio de instrucciones

Como hemos comentado, en este proyecto hemos trabajado con un pequeño repertorio de las instrucciones de C++, puesto que abarcar toda su sintaxis era inviable. Es por esto que trabajamos con instrucciones básicas y el tipo de números enteros.

Si esta aplicación se quisiese usar como un depurador completo, sería necesario contemplar toda la sintaxis de C++ y todos los tipos de variables.

### Otros sistemas operativos

Una de las limitaciones que tuvimos a la hora de desarrollar la aplicación es la instalación del compilador Clang en el sistema operativo de Windows, por tanto nos vimos obligados a limitarnos a desarrollar en Linux. Esta incompatibilidad limita la portabilidad de la herramienta y por tanto el número de usuarios que podrían utilizarla.

Si la apliacación quisiese abarcar a una mayor comunidad de usuarios sería necesario investigar el modo de compilar la herramienta de Clang en otros sistemas operativos como Windows o MacOSX.

## Aplicacion web

En referencia a la sección anterior, una posible ampliación sería exportar esta aplicación al ámbito web. De este modo, no sería necesario instalación de librerías u otros programas que tengan incompatibilidades con el sistema operativo, y por otro lado los usuarios no sólo se limitarían a los que conociesen la aplicación por la Universidad sino que la comunidad potencial es global.

# Apéndice A

# Manual de usuario

## A.1. Requisitos previos

Como ya hemos indicado en las secciones de la memoria, nuestro proyecto actualmente sólo funciona en sistemas operativos LINUX, por tanto en lo siguiente supondremos que se trabaja bajo este sistema operativo.

Por otro lado para poder ejecutar el sistema que hemos desarrollado será necesario cumplir una serie de requisitos software previos. En concreto deberemos tener instalados los siguientes componentes:

- Java Runtime Environment o JRE de 32 bits, versión 1.6 o superior.
  Para ello desde la propia página recomiendan, en lo relativo a requisitos hardware, contar con un Pentium 2 a 266 MHz o un procesador más rápido con al menos 128 MB de RAM física.
  Enlace de descarga: `http://www.java.com/es/download/`

- Clang. Al comienzo del proyecto la última versión disponible era la 3.5, por lo que esta fue la que se utilizó.
  Para poder compilar nuestra herramienta de generación dle AST2XML, cuyos pasos detallados de instalación comentamos en la próxima sección, es necesario tener instalado Clang en nuestro sistema. Puesto que la instalación y requisitos ya están detallados en la propia web de Clang no vamos a detenernos a comentarlos uno a uno.

  - Requisitos LLVM System
    Enlace de requisitos: `http://llvm.org/docs/GettingStarted.html#requirements`
  - Python. Para compilar Clang es necesario tener instalado Python en nuestro sistema.

Enlace de descarga: `https://www.python.org/download`

- Compilacion de Clang
  Enlace con los pasos detallados `http://clang.llvm.org/get_started.html`

## A.2. Instalación Clang

# Bibliografía

*Y así, del mucho leer y del poco dormir,*
*se le secó el celebro de manera que vino*
*a perder el juicio.*

Miguel de Cervantes Saavedra

*–¿Qué te parece desto, Sancho? – Dijo Don Quijote –*
*Bien podrán los encantadores quitarme la ventura,*
*pero el esfuerzo y el ánimo, será imposible.*

*Segunda parte del Ingenioso Caballero*
*Don Quijote de la Mancha*
*Miguel de Cervantes*

*–Buena está – dijo Sancho –; fírmela vuestra merced.*
*–No es menester firmarla – dijo Don Quijote–,*
*sino solamente poner mi rúbrica.*

*Primera parte del Ingenioso Caballero*
*Don Quijote de la Mancha*
*Miguel de Cervantes*