41ª *Jornadas de Informática e Investigación Operativa*

FOLST: Una Herramienta Didáctica para la Lógica de Predicados de Primer Orden

Autores **Gervasoni, Luciano**
Maggiori, Emmanuel

{lgervasoni, emaggiori}@alumnos.exa.unicen.edu.ar

Directores **Felice, Laura**
Mauco, Virginia
Ferrante, Enzo

{lfelice, vmauco}@exa.unicen.edu.ar
eferrante@alumnos.exa.unicen.edu.ar

Categoría **Trabajos de Cátedra**

Área **Ambientes de programación, Algoritmos, Lenguajes, Modelos Lógica, Inteligencia Artificial y Sistemas Inteligentes**

Asignaturas **Análisis y Diseño de Algoritmos I**
Ciencias de la Computación II

Segundo año de la carrera de Ingeniería de Sistemas,
Facultad de Ciencias Exactas,
Universidad Nacional del Centro de la Provincia de Buenos Aires,
Tandil

FOLST: Una Herramienta Didáctica para la Lógica de Predicados de Primer Orden

Resumen

FOLST es una herramienta didáctica, visual e interactiva que fue diseñada como soporte para el proceso de enseñanza/aprendizaje de semántica en la Lógica de Predicados de Primer Orden. La herramienta es fácil de usar y, mediante una interfaz gráfica amigable e intuititva, permite a los usuarios analizar el valor de verdad de fórmulas en modelos definidos a partir de dos *frames* provistos por la herramienta. FOLST es software libre y está implementada en el lenguaje de programación C++.

Se desarrolló como Trabajo Final para dos materias que se dictan en el segundo año de una carrera de Informática, que involucran contenidos en lógica y en análisis y diseño de algoritmos.

1. Introducción

Los cursos básicos de Lógica se dictan en la mayoría de las carreras de Informática. En estos cursos, los estudiantes deben resolver gran cantidad de ejercicios para adquirir práctica en el manejo de formalismos. El uso de herramientas educativas puede ser un gran aporte en este sentido. Por esto, se consideró desarrollar como Trabajo Final para dos materias (ambas se dictan en el primer semestre del segundo año de una carrera de Informática) que involucran contenidos en lógica y en análisis y diseño de algoritmos, una herramienta didáctica que fuera intuitiva en su uso y cuya notación se correspondiera con aquella utilizada en el curso de Lógica.

La Lógica de Predicados de Primer Orden (LdPPO) [3] [6] incluye predicados y funciones (que se aplican sobre un dominio), para definir un modelo o interpretación. El valor de verdad de una fórmula depende de la definición del modelo en el cual se la evalúa, lo que le agrega dificultad en comparación con la lógica proposicional. Existen estrategias que buscan, por ejemplo, encontrar contradicciones en las fórmulas. Para este tipo de problemas, se han implementado herramientas didácticas, como [7] [8]. Sin embargo, evaluar semánticamente una fórmula, en general, resulta más difícil para los estudiantes debido a la posibilidad de definir infinitos modelos arbitrarios. [2] [4] son dos herramientas pensadas como soporte en estos temas.

Para complementar este proceso de enseñanza/aprendizaje de semántica en LdPPO, se desarrolló FOLST (First Order Logic Semantic Tutor), que es una herramienta didáctica, visual e interactiva que permite la evaluación de fórmulas en modelos construidos por el usuario usando dos *frames* (Granja y Mundo) provistos por la herramienta. De esta manera, se puede experimentar el significado de la LdPPO en situaciones reales, conociendo sus alcances y limitaciones. La herramienta fue implementada usando el lenguaje de programación C++ y la librería Qt [10] para la interfaz gráfica. También, ha sido liberada bajo la licencia GNU GPL v3.0 [1].

En la Sección 2 se describe el diseño global de la herramienta. La Sección 3 contiene una descripción de las principales funcionalidades de FOLST. Finalmente, en la Sección 4 se presentan las conclusiones sobre el trabajo realizado.

2. Diseño e Implementación

Se buscó implementar una herramienta interactiva y visual. Se utilizó el paradigma Orientado a Objetos, definiendo un conjunto de Tipos de Datos Abstractos (TDA), y programando en el lenguaje C++. En la arquitectura de la aplicación, se pueden distinguir dos niveles:

Nivel Lógico Este nivel define los TDA para la representación de fórmulas y su evaluación semántica en un modelo determinado. Los TDA proveen una interfaz para manipular elementos de la LdPPO, y definen algunos métodos básicos para obtener información útil, por ejemplo el TDA Formula define una de las funcionalidades más importantes de FOLST que es la determinación de la validez de una fórmula en un modelo dado.

Nivel de Presentación Este nivel contiene el código para la interfaz gráfica de la herramienta, que debía ser ágil e intuitiva. Se empleó la librería Qt [10] que simplifica el diseño y permite la portabilidad a distintas plataformas. En particular se hizo uso del componente *QGraphicsView* para representar los *frames* Mundo y Granja implementados en la herramienta.

En las siguientes secciones se describen algunos aspectos del Nivel Lógico.

2.1. Tipos de Datos Abstractos

El problema clave que se debía resolver era la evaluación semántica de fórmulas en LdPPO. Para esto se utilizó un enfoque Orientado a Objetos, de manera tal de definir TDA's [9] que se correspondieran con las distintas entidades participantes en este tipo de lógica. Por la naturaleza recursiva de la definición de las fórmulas, se utilizó un patrón que usa la composición para representarlas (Figura 1). De esta manera, las fórmulas se representan como una estructura arborescente (Figura 2).

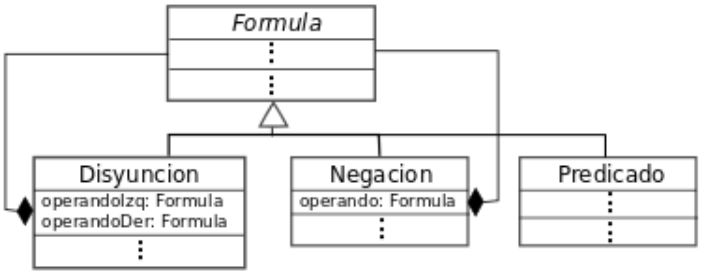

Figura 1: Patrón utilizado para representar fórmulas

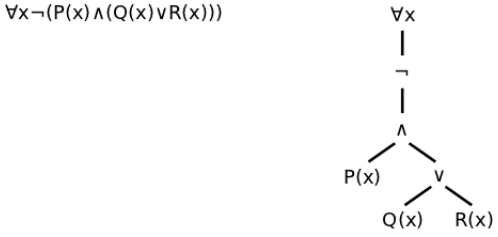

Figura 2: Ejemplo de una fórmula y su árbol asociado

Adicionalmente, se definió una clase abstracta llamada *Modelo*, que define la interfaz para comunicarse con los métodos de análisis semántico de fórmulas. De esta manera, se permite la creación de múltiples modelos. Estos deben heredar y definir a la clase *Modelo* y deben cumplir las tareas de mantener el estado del modelo, responder por la validez o no de un predicado, evaluar una función y devolver el dominio de esa instancia del modelo (Figura 3). Así, se independizó completamente el modelo en cuestión del análisis semántico de las fórmulas, y se permite de esta manera la fácil incorporación de nuevos modelos en la herramienta.

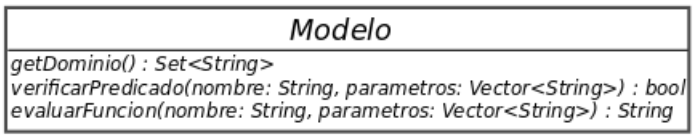

Figura 3: Clase *Modelo*

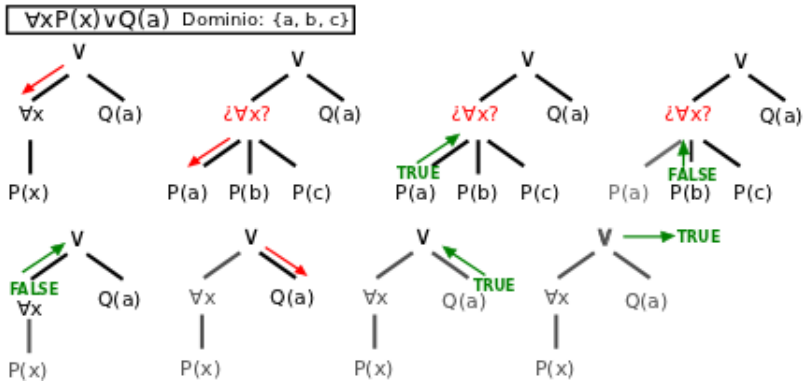

Figura 4: Evaluación semántica de una fórmula cuantificada

2.2. Algoritmo de Evaluación Semántica

La estrategia de evaluación semántica consiste en hacer un recorrido post-orden del árbol asociado a una fórmula. En este recorrido, se accede a las hojas, se las evalúa en el modelo, y se empieza a hacer un recorrido hacia atrás en el cual se van aplicando las operaciones hasta llegar al nodo raíz, de manera recurrente. Cuando un nodo contiene un cuantificador, se repite lo que queda del recorrido varias veces. Es decir, se itera dependiendo del cuantificador, con distintas instancias para la variable cuantificada, hasta poder determinar la validez de la fórmula afectada por el cuantificador (Figura 4).

Un predicado en LdPPO se define con un nombre y una lista de términos, que pueden ser funciones, variables o constantes. Cualquier instancia de la clase *Modelo* tiene la capacidad de evaluar predicados, dado un nombre y una lista de constantes. Por ejemplo, puede decidir la validez de Predicado(a,b). Sin embargo, la evaluación de, por ejemplo, Predicado(a,f(b)) no es tarea del modelo. Es el algoritmo de evaluación semántica quien deberá preguntar primero al modelo por la constante que se asocia a f(b), supóngase c, y luego consultar por Predicado(a,c).

El algoritmo resuelve este tipo de casos de manera recursiva, dado que cada función puede tener también una lista de términos que involucren incluso otras funciones, y así sucesivamente. De esta manera se construye la lista de argumentos del predicado, para luego efectuar la consulta al modelo (Figura 5).

2.2.1. Complejidad temporal del algoritmo de recorrido del árbol

Como parte del trabajo, se analizó la complejidad temporal del algoritmo de recorrido del árbol que representa una fórmula. En este caso, se buscó definir la complejidad en función de la cantidad de operadores lógicos. La presencia de cuanti-

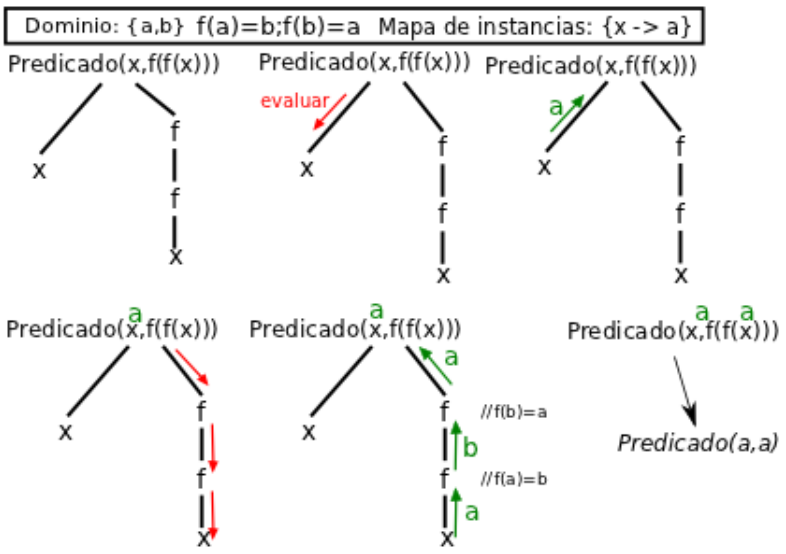

Figura 5: Evaluación de predicados

ficadores altera considerablemente la cantidad de operaciones, por iterar en función del dominio. Entonces, para simplificar los cálculos, se decidió crear un árbol genérico que represente el peor de los casos en función de la cantidad de operadores, para lo cual se ubicaron todos los cuantificadores en la raíz del árbol, seguidos por los operadores de la lógica proposicional y, finalmente, los predicados en las hojas (Figura 6).

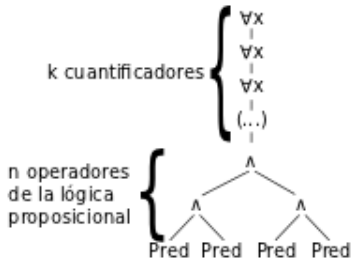

Figura 6: Árbol que representa el peor caso en costo temporal

Se puede ver que recorrer el subárbol correspondiente a los operadores de la lógica proposicional, tiene un costo de n, siendo n la cantidad de operadores. Es

decir, para resolver una fórmula no cuantificada, alcanza con visitar una vez cada nodo. Si se añade un cuantificador en la raíz de este árbol, en el peor caso, se realizará el recorrido anterior d veces (siendo d el tamaño del dominio). Esto es, un total de dn veces. En general, se realizan $d^k n$ pasos, siendo k la cantidad de cuantificadores.

La complejidad temporal del recorrido resulta ser, entonces, del orden $O(d^k n)$.

Al analizar el resultado se advierte que el mayor peso en la complejidad está dado por la cantidad de cuantificadores. Justamente, son los cuantificadores la novedad que aporta la LdPPO a la lógica proposicional, y es curioso notar que sean los que introducen mayor complejidad en el análisis semántico de las fórmulas.

2.3. Analizador Léxico-Sintáctico

Para la creación del árbol correspondiente a una fórmula a partir de una cadena de texto, se generó un *parser* mediante Bison Parser Generator [5] y para la identificación de subsecuencias en la cadena se empleó Flex [11].

Se debieron definir los *tokens* para identificar variables o constantes, funciones y predicados (IDVAR, IDFUNC, IDPRED). Los tres debían ser distinguibles entre sí. Se decidió que las variables deberían identificarse con una serie de minúsculas (por ejemplo, "x") y las funciones con una serie de mayúsculas (por ejemplo, "ELMASALEJADO") . Los predicados debían contar con un primer caracter en mayúscula y el que sigue en minúscula (por ejemplo, EsCapital). Para los operadores lógicos se decidió utilizar los mismos caracteres que se emplean en el curso de lógica y en la bibliografía, y se permitió también la posibilidad de utilizar caracteres alternativos (como '@' para '∀') para facilitar el uso desde teclado. La gramática que define las fórmulas de la LdPPO se describe con el BNF de la Figura 7.

comienzo := cond //para realizar acciones al principio
cond := disy → cond | disy
disy := conj ⋁ disy | disy
conj := factor ⋀ conj | factor //factor: unidad cuantificada o negada
factor :=∀ IDVAR factor | ∃ IDVAR factor | ¬ factor | (cond) | pred
pred := IDPRED (terms) //terms representa una lista de términos
terms := term | terms , term
term := IDVAR | IDFUNC (terms)

Figura 7: Gramática Libre del Contexto empleada en el análisis sintáctico

3. Características de FOLST

Dado que FOLST ha sido diseñada con fines didácticos, es importante destacar las funcionalidades que incluye orientadas al proceso de enseñanza/aprendizaje de semántica en la LdPPO. Durante este proceso, los alumnos pueden tener que enfrentarse con los siguientes problemas/cuestiones:

- decidir si una fórmula es sintácticamente correcta;
- evaluar la validez de una fórmula en un modelo;
- crear nuevos modelos o modificarlos para comparar los cambios en la validez de las fórmulas en ellos;
- determinar si una fórmula es lógicamente válida (válida en todo modelo), contradictoria (falsa en todo modelo), o satisfacible (válida en un modelo pero falsa en otro).

FOLST ofrece soporte didáctico para realizar todas estas tareas, permitiendo definir múltiples modelos y fórmulas a la vez, guardar y cargar modelos y fórmulas, y dando información de errores en la definición de las fórmulas.

En las próximas secciones se describen con un poco más de detalle algunas de las funcionalidades de FOLST.

3.1. *Frames* Mundo y Granja

Hasta el momento, la herramienta cuenta con la implementación de dos *frames*, Granja y Mundo, que permiten la creación de diferentes modelos.

El *frame* Granja (Figura 8) consiste en una imagen de una granja en la que distintos animales (por ejemplo, vacas y gallinas) se pueden agregar en distintos lugares (el corral, el bosque, etc.). Cada animal tiene atributos (su especie, ubicación y si está durmiendo) y predicados que permiten formalizar información real en este contexto. El *frame* provee once predicaros unarios, como *EsVaca(x)*, *EstaEnElPasto(x)* y *EstaDurmiendo(x)*, y dos binarios, *MismoLugar(x,y)* y *MismaEspecie(x,y)*. Además, se incorporó una función que, dado un animal, retorna el más cercano a éste (*ELMASCERCANO(x)*).

El *frame* Mundo (Figura 9) provee un mapa dividido en continentes sobre el cual se pueden agregar ciudades (capitales o no) que se pueden conectar. Se definieron seis predicados unarios, como *EsCapital(x)* o *EstaEnAmerica(x)*, y cinco binarios, como *MismoContinente(x,y)* y *SePuedeLlegar(x,y)*. La función *LAMASALEJADA(x)* devuelve, para una ciudad, la más alejada.

FOLST permite trabajar con varios modelos a la vez y su manipulación es interactiva y gráfica.

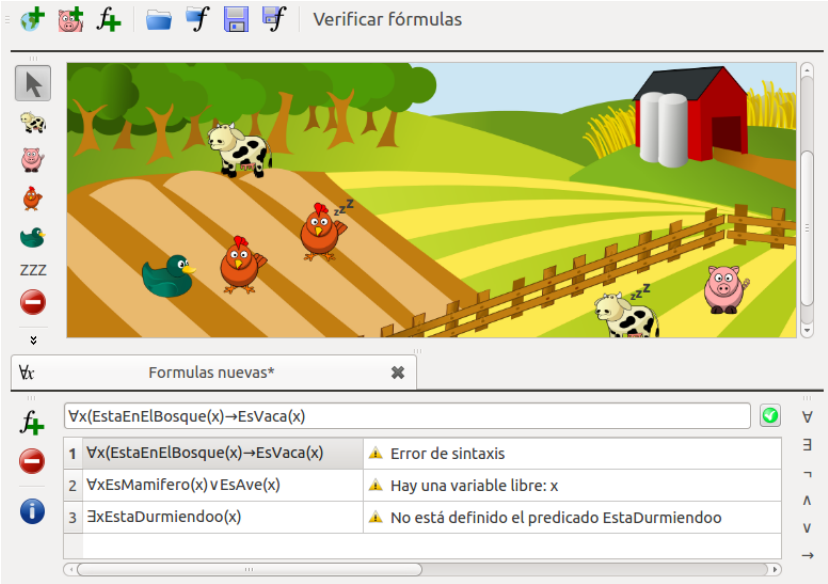

Figura 8: Modelo definido por el usuario en el *frame* Granja

3.2. Edición de Fórmulas

Las fórmulas se pueden definir mediante un editor e independientemente del modelo en el cual se las quiera evaluar. Se facilitó mediante botones la inserción de los operadores lógicos para que las fórmulas puedan tener el mismo aspecto que en la bibliografía.

Al evaluar fórmulas se realiza un análisis sintáctico y se proporciona información de los errores, para que los usuarios los puedan identificar fácilmente y corregirlos. Como se ve en la Figura 8 los errores pueden ser independientes del modelo en el que se esté evaluando la fórmula (por ejemplo, la presencia de una variable libre y la falta de un paréntesis) o dependientes del modelo (por ejemplo, que un predicado no esté definido).

3.3. Evaluación de Fórmulas

Para cada fórmula del editor, FOLST calcula su valor de verdad en el modelo que esté en primer plano, retornando “Válida” o “Falsa” en el modelo. El usuario puede cambiar el modelo, por ejemplo agregando o moviendo ciudades en un modelo del *frame* Mundo, y recalcular los valores de verdad. En la Figura 9 todas las fórmulas son sintácticamente correctas y entonces FOLST muestra sus valores de verdad.

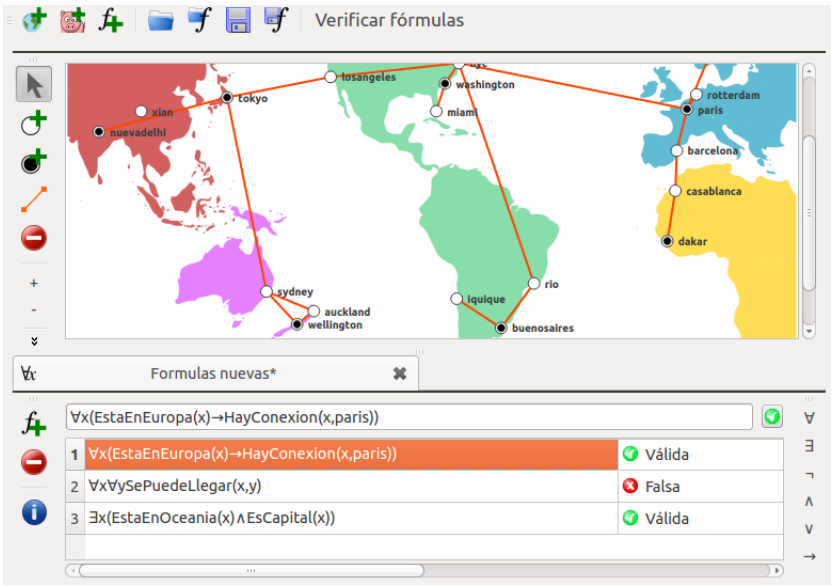

Figura 9: Modelo definido por el usuario en el *frame* Mundo

4. Conclusiones

Este trabajo permitió integrar los contenidos aprendidos en dos materias de distinta naturaleza, lo que lo convierte en una herramienta aplicada. Sirvió tanto para hacer uso de las técnicas de programación, como para afianzar los conocimientos en Lógica. Se trató también de la primera experiencia en la implementación de una interfaz gráfica, lo cual fue muy positivo haciendo uso del framework Qt.

Se buscó que la herramienta desarrollada fuera didáctica, para ayudar a comprender el alcance y el significado de la LdPPO. Fue desarrollada por estudiantes para estudiantes, lo cual lo convierte en algo motivador en el proceso de aprendizaje. Actualmente se la está utilizando en el curso de lógica como una herramienta complementaria. FOLST se enfocó en la apariencia de las fórmulas, para que fueran idénticas a aquellas que se usan en clase, y se la orientó para permitir el trabajo con múltiples modelos y fórmulas a la vez, enfatizando el hecho de que en LdPPO el valor de verdad de una fórmula depende del modelo en el que se la evalúe.

El diseño de la herramienta se pensó para que ésta fuera extensible, pudiéndose incorporar modelos nuevos de manera fácil, simplemente definiendo sus predicados y funciones, y sin tener que trabajar sobre el análisis sintáctico o sobre el algoritmo de evaluación semántica. La idea es seguir extendiendo la herramienta y alentar a los estudiantes a que lo hagan también. Para esto se trabajó íntegramente con software

libre y con tecnologías independientes de la plataforma. Al ser su código abierto, los estudiantes pueden analizar su funcionamiento, lo que es más que interesante debido a que se desarrolló utilizando las mismas tecnologías que ellos están aprendiendo en el momento: es muy deseable que los estudiantes puedan ver aplicaciones tangibles del lenguaje de programación y del paradigma que están aprendiendo. Se escribió también una guía para extender la herramienta, disponible para los estudiantes.

La posibilidad de guardar y cargar archivos facilita la construcción de modelos complejos y el intercambio de ellos, así como permite al docente compartir ejemplos. Se escucharán las recomendaciones de los estudiantes para mejorar la herramienta y que sea de la mayor utilidad posible en su aprendizaje.

El objetivo de la herramienta es, entonces, permitir al estudiante experimentar con la lógica en situaciones de la vida cotidiana, a pesar de la naturaleza abstracta de ésta, y conocer sus alcances y limitaciones. Este conocimiento es deseable antes y durante el aprendizaje de algoritmos y técnicas de evaluación de fórmulas de LdPPO.

Referencias

[1] GNU General Public License, version 3.0. http://www.gnu.org/licenses/gpl-3.0.html, Accessed Abril 2012.

[2] D. Baker-Plummer, J. Barwise, and J. Etchemendy. Tarski's World. University of Chicago Press, 2007.

[3] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer Verlag, 2001.

[4] Departamento de Ciencia de la Computación e Inteligencia Artificial de la Universidad de Alicante. Moros y cristianos. http://www.dccia.ua.es/logica/MyC.

[5] Free Software Foundation. Bison - GNU Parser Generator Manual. http://www.gnu.org/s/bison/.

[6] J. Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[7] M. V. Mauco and E. Ferrante. Clausula: A Didactic Tool to Teach First Order Logic. In *Information Systems Education Conference*, volume 26, 2009.

[8] M. V. Mauco, L. Moauro, and L. Felice. Una Herramienta Didáctica para la Enseñanza de Lógica de Predicados de Primer Orden. CIESC, 2010. Paraguay.

[9] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall, 1997.

[10] Nokia. Online Reference Documentation. http://doc.qt.nokia.com/.

[11] Flex Project. Lexical Analysis With Flex. http://flex.sourceforge.net/manual/.