Plan de Proyecto

Propuesta de Tema de Tesis

Título Tentativo

Joy: *Just Optimized Yielder* - Generación Automática de Pruebas para Gramáticas en Haskell

Justificación

El desarrollo de lenguajes de programación es una tarea compleja que requiere herramientas eficientes para probar y validar la corrección de las gramáticas. Actualmente, muchas de las herramientas existentes para la generación de expresiones dada una gramática son generales y no están optimizadas para un enfoque declarativo, lo que dificulta su integración en flujos de trabajo modernos; por ejemplo, la herramienta *yagg*[9] cuenta con una línea de comandos para generar expresiones aleatorias dada una gramática, pero no sólo fue programado imperativamente, sino también las configuraciones de las cadenas están limitadas, y no es posible especificar la creación de cadenas que sean correctas semánticamente, lo que dificulta su uso para realizar pruebas en la implementación de un lenguaje de programación; también está la herramienta *DGL*[10] que tiene la capacidad de determinar el peso para determinar las frecuencias de las reglas de producción, y una documentación extensa para extender la funcionalidad del programa, sin embargo al ser un programa por separado, se dificulta su uso dentro de un entorno declarativo, pues en él se intenta prescindir lo más posible de los efectos secundarios, y su uso para realizar pruebas implicaría tener una constante comunicación con el Sistema Operativo, lo cual no es ideal.

Este proyecto propone la herramienta **Joy: *Just Optimized Yielder***, la cual permite a los desarrolladores definir gramáticas de manera declarativa usando *EBNF* (*Extended Backus-Naur Form*) y generar automáticamente herramientas de prueba para el desarrollo de un lenguaje de programación en *Haskell*. Al utilizar un enfoque monádico[1] para el análisis sintáctico y la generación de pruebas, **Joy** ofrece una forma eficiente y modular de validar lenguajes de programación. Además, al integrar generadores basados en *QuickCheck*[11], se garantiza que las pruebas sean robustas y cubran una amplia gama de casos.

En el ámbito académico, Joy pretende contribuir al estudio de lenguajes formales y programación funcional, integrando técnicas como analizadores sintácticos monádicos y generadores de pruebas basados en propiedades (*QuickCheck*), fomentando la investigación en métodos eficientes para el análisis sintáctico y la generación automática de pruebas. Y en el ámbito práctico, Joy pretende simplificar el desarrollo de lenguajes de programación al automatizar la creación de herramientas de validación, reduciendo el tiempo y esfuerzo requeridos para probar gramáticas y mejorando la calidad de los lenguajes mediante pruebas robustas. Su enfoque declarativo y modular lo hace adaptable a diversos contextos, desde lenguajes de dominio específico hasta lenguajes de propósito general, posicionándolo como una solución versátil y valiosa tanto para la industria como para la academia.

Objetivos

Objetivo General

Desarrollar una herramienta llamada **Joy: *Just Optimized Yielder*** que permita a los desarrolladores de lenguajes de programación generar automáticamente casos de prueba para validar gramáticas en formato *EBNF*.

Objetivos Específicos

1. Implementar un analizador sintáctico en *Haskell* que interprete archivos `.joy` con gramáticas descritas en formato *EBNF*. Éste formato a diferencia del estándar utilizado para el generador de analizadores sintácticos *YACC*, tendría el objetivo de ser el punto de partida para el desarrollo, teniendo una estructura de fácil comprensión, y pueda funcionar así mismo como documentación para el lenguaje de programación.

2. Generar automáticamente código de *Haskell* que incluya generadores de pruebas para cada regla de producción definida en la gramática. Estos generadores seguirían un enfoque basado en propiedades.

3. Integrar generadores basados en *QuickCheck* para garantizar la robustez de las pruebas.

[1] Las mónadas son una estructura algebráica utilizada para manejar secuencias de operaciones que involucran contextos computacionales o efectos secundarios, como manejo de errores, estado, entrada/salida, o cálculos no determinísticos. En este proyecto principalmente se utilizaría para tener control del estado del programa durante el análisis sintáctico, así como describir la manipulación del flujo de la información generada aleatoriamente.

4. Permitir la configuración de parámetros adicionales para restringir tipos de datos, determinar las reglas de producción a usar en los generadores, o determinar el tamaño de la longitud del árbol.
5. Opcional: Implementar la capacidad de evaluar cadenas generadas mediante un intérprete configurable.
6. Documentar y validar la herramienta mediante casos de estudio concretos.

Descripción de los casos de estudio

Para validar la efectividad de *Joy*, se proponen los siguientes casos de estudio:

1. *Lenguaje de Expresiones Aritméticas*
   - *Descripción:* Un lenguaje simple que incluye operaciones básicas como suma, resta, multiplicación y división.
   - *Objetivos:* Evaluar que los generadores de pruebas produzcan una cobertura representativa de las expresiones aritméticas permitidas por la gramática.
2. *Expresiones básicas en LISP*
   - *Descripción:* Un subconjunto del lenguaje *LISP* con funcionalidades simples como la definición de variables mediante expresiones `let` y la aplicación de funciones.
   - *Objetivo:* Validar que los generadores produzcan expresiones correctas tanto sintácticamente como semánticamente con ayuda de un intérprete externo, como *Racket*.
3. *Programación implerativa*
   - *Descripción:* Un lenguaje completo, que cuente con la capacidad de generar procedimientos descritos de forma imperativa, expresiones `if`, ciclos `while`, `for`, definición de métodos, etc...
   - *Objetivo:* Comprobar que el sistema desarrollado es capaz de generar expresiones de un lenguaje con más especificaciones, y verificar la utilidad de los generadores para desarrollar pruebas específicas en el desarrollo éste lenguaje.

Metodología

1. *Investigación y Análisis*:
   - Revisión del estado del arte en herramientas de generación de pruebas y análisis sintáctico.
   - Estudio de bibliotecas de *Haskell* relevantes (*QuickCheck*,*State*,etc...).
   - Estudio de bibliotecas similares en otros lenguajes.
2. *Diseño*:
   - Definición de la estructura sintáctica de los archivos `.joy`, que a diferencia de otros como el del lenguaje de especificación de *Happy*[12], seguirá el estandar ISO/IEC 14977 de *EBNF* para una mejor comprensión de la gramática.
   - Diseño del analizador sintáctico basado en *parsers* monádicos desde cero.
   - Diseño de los generadores de pruebas y su integración con *QuickCheck*
3. *Implementación*:
   - Desarrollo del analizador sintáctico en *Haskell*.
   - Implementación de la traducción automática de *EBNF* a generadores aleatorios en *QuickCheck*.
   - Integración de parámetros adicionales para restricciones en los generadores, como longitud de las cadenas generadas, exclusión de reglas de producción, pesos que determinen la frecuencia de las reglas de producción, entre otras.

- Implementación opcional de la evaluación mediante intérpretes externos.

4. *Validación*:
   - Aplicación de los casos de estudio para validar la funcionalidad de **Joy**
   - Pruebas unitarias y de integración por cada componente
     - Comprobar que las expresiones generadas siguen la estructura sintáctica que se definió.
     - Verificar que las expresiones cumplen con las restricciones que se especificaron al generador.
     - Corroborar que se crean expresiones semánticamente correctas mediante el uso de intérpretes externos.
     - Probar la extensibilidad de las pruebas y su integración con *QuickCheck*.
     - Comparar el tiempo de compilación con el de otras herramientas similares (e.g. *Happy*).
5. *Redacción del Trabajo de Tesis*
   - Redacción de la documentación técnica y del usuario.
   - Preparación de ejemplos y tutoriales.
6. *Evaluación y Mejora*:
   - Revisión de los resultados.
   - Identificación de áreas de mejora y optimización.

Resultados Esperados

Se espera que **Joy** sea una herramienta funcional y eficiente que cumpla con los siguientes resultados:

- Generación automática de expresiones a partir de gramáticas descritas en *EBNF*.
- Integración exitosa con *QuickCheck* para la generación de casos de prueba robustos.
- Capacidad para manejar gramáticas libres del contexto, como lenguajes de programación de propósito específico.
- Documentación clara y ejemplos que faciliten su uso por parte de otros desarrolladores.
- Validación exitosa de cada componente, mediante los casos de estudio propuestos y las pruebas unitarias mencionadas.

Además, se espera que **Joy** sirva como una contribución significativa al campo de desarrollo de lenguajes de programación, promoviendo el uso de técnicas declarativas y funcionales.

Índice Propuesto

El documento de la tesis seguirá la siguiente estructura:

1. **Introducción**
   - Contexto y motivación.
   - Objetivos del proyecto.
   - Antecedentes: analizadores sintáticos monádicos, proceso del compilador, *QuickCheck* y sus módulos.
2. **Estado del Arte**
   - Herramientas existentes para generación de pruebas.
   - Análisis sintáctico en *Haskell*.

3. **Diseño de Joy**
    - Descripción de la herramienta.
    - Especificación del formato `.joy`
    - Arquitectura del sistema.
4. **Implementación**
    - Analizador sintáctico.
    - Generadores de pruebas.
    - Integración con *QuickCheck*.
    - Funcionalidades opcionales.
5. **Casos de Estudio**
    - Lenguaje de expresiones aritméticas.
    - Subconjunto del lenguaje *LISP*.
    - Lenguaje imperativo de propósito específico.
6. **Resultados y Discusión**
    - Evaluación de la herramienta.
    - Limitaciones y áreas de mejora.
7. **Conclusiones y Trabajo Futuro**
    - Conclusiones del proyecto.
    - Posibles extensiones y mejoras.
8. **Referencias**
    - Bibliografía y recursos utilizados.
9. **Anexos**
    - Código fuente.
    - Repositorio con el contenido del proyecto.
    - Manual de usuario.

Referencias

[1] Lipovaca, M. (2011). *Learn you a Haskell for great good! A beginner's guide* (1st ed.). No Starch Press.

[2] Hutton, G. (2007). *Programming in Haskell* (3rd ed.). Cambridge University Press.

[3] Pedraza, D. (2018). *Teoría de categorías y programación funcional* (Trabajo de titulación). Universidad de Sevilla.

[4] Lee, K. D. (2017). *Foundations of programming languages*. Springer.

[5] Pierce, B. C. (2002). *Types and programming languages*. MIT Press.

[6] Thain, D. (2020). *Introduction to compilers and language design* (2nd ed.). University of Notre Dame.

[7] Louden, K. C. (1997). *Compiler construction: Principles and practice*. PWS Publishing Company.

[8] International Organization for Standardization. (1996). *Information technology — Syntactic metalanguage — Extended BNF* (ISO/IEC 14977:1996). Recuperado de https://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[9] Coppit, D. (2018, julio). *Random generator manual*. Recuperado de https://metacpan.org/dist/yagg/view/random_generator

[10] Maurer, H. A. (n.d.). *DGL manual*. Recuperado de https://cs.baylor.edu/~maurer/aida/dgl-source/documentation/dglmanul.pdf

[11] Claessen, K., & Hughes, J. (n.d.). *QuickCheck manual*. Recuperado de https://www.cse.chalmers.se/~rjmh/QuickCheck/manual.html

[12] Happy. (n.d.). *Happy documentation*. Recuperado de https://haskell-happy.readthedocs.io/en/latest/