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.

Se propone la herramienta **Joy: *Just Optimized Yielder*** como una solución que permita a los desarrolladores describir gramáticas de manera declarativa usando *EBNF* (*Extended Backus-Naur Form*) y generar automáticamente herramientas de prueba 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*, se garantiza que las pruebas sean robustas y cubran una amplia gama de casos.

Esta herramienta no sólo simplifica el proceso de desarrollo de lenguajes, sino que también promueve el uso de técnicas funcionales y declarativas.

Objetivos

Objetivo General

Desarrollar una herramienta llamada **Joy: *Just Optimized Yielder*** que permita a los desarrolladores de lenguajes de programación generar automáticamente herramientas de prueba a partir de una descripción declarativa de la gramática en formato *EBNF*

Objetivos Específicos

1. Implementar un analizador sintáctico en *Haskell* que interprete archivos `.joy` con gramáticas descritas en formato *EBNF*.
2. Generar automáticamente código de *Haskell* que incluya generadores de pruebas para cada regla de producción definida en la gramática.
3. Integrar generadores basados en *QuickCheck* para garantizar la robustez de las pruebas.
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 mendiante casos de estudio concretos.

[1] Las mónadas son un patrón de diseño comúnmente utilizados en lenguajes de programación funcionales que permiten una abstracción puramente funcional de tareas que involucrarían el uso de efectos secundarios. En este proyecto principalmente se utilizaría para tener control del estado del programa durante el análisis sintáctico, así como describir la manipulación el flujo de la información generada aleatoriamente.

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:* Verificar que los generadores de pruebas cubran la mayor cantidad de combinaciones posibles de expresiones aritméticas.

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, y que estas puedan ser evaluadas en un intérprete 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...).

2. *Diseño*:

   - Definición de la estructura sintáctica de los archivos `.joy`
   - Diseño del analizador sintáctico basado en *parsers* monádicos.
   - 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.
   - Implementación opcional de la evaluación mediante intérpretes externos.

4. *Validación*:

   - Pruebas unitarias y de integración por cada componente.

- Aplicación de los casos de estudio para validar la funcionalidad de **Joy**

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 pruebas a partir de gramáticas descritas en *EBNF*.
- Integración exitosa con *QuickCheck* para la generación de casos de prueba robustos.
- Capacidad para manejar gramáticas complejas con restricciones específicas.
- Documentación clara y ejemplos que faciliten su uso por parte de otros desarrolladores.
- Validación exitosa mediante los casos de estudio propuestos.

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.
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*.
- Funcionalidads opcionales.

5. **Casos de Estudio**

   - Lenguaje de expresiones aritméticas.
   - Expresiones básicas en *LISP*.

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.