Relatório de Projeto: Compilador SL – Trabalho Prático 2

Thalles Felipe e Júlia Gonzaga
21.2.4130 e 21.2.4115
BCC328 - Construção de Compiladores I - DECOM/UFOP

14 de fevereiro de 2026

Resumo

Este relatório apresenta a evolução do compilador da linguagem SL (Simple Language), desenvolvido em Haskell, contemplando as funcionalidades implementadas no Trabalho Prático 2 da disciplina BCC328. Nesta etapa, o projeto foi estendido para incluir análise semântica completa, inferência de tipos, interpretação de programas e uma AST parametrizada por anotações. O compilador agora suporta verificação de tipos e escopo, operadores pós-fixos, execução de programas SL via interpretador e um pretty printer com garantia de round-trip. A implementação reforça a robustez do front-end e consolida a linguagem como semanticamente bem definida e executável.

Sumário

1 Introdução

Este trabalho prático dá continuidade ao desenvolvimento de um compilador para a linguagem SL, iniciado no Trabalho Prático 1. Enquanto a primeira entrega concentrou-se

na análise léxica, sintática e na construção da AST, esta segunda etapa tem como foco principal a **análise semântica** e a **execução de programas** por meio de um interpretador.

A linguagem SL é uma linguagem imperativa estaticamente tipada, projetada para fins didáticos, oferecendo suporte a funções, estruturas de controle, arranjos, registros (`struct`), polimorfismo paramétrico (`forall`) e tipos função. A Entrega 2 amplia significativamente o compilador, tornando-o capaz de detectar erros semânticos, inferir tipos ausentes e executar programas corretamente.

2 Metodologia

O desenvolvimento seguiu uma abordagem incremental e modular, reaproveitando a base construída na Entrega 1. A arquitetura do compilador foi estendida para incluir novas fases após a construção da AST, mantendo separação clara de responsabilidades entre módulos.

As principais extensões realizadas nesta etapa foram:

- Implementação da análise semântica com verificação de tipos e escopo;
- Introdução de inferência de tipos para parâmetros e variáveis sem anotação explícita;
- Criação de um interpretador para execução direta de programas SL;
- Parametrização da AST com anotações semânticas (tipos e posições);
- Suporte a operadores pós-fixos (`i++` e `i-`);
- Garantia de round-trip no pretty printer.

2.1 Análise semântica

A análise semântica percorre a AST validando regras que não podem ser expressas apenas pela gramática. Entre as verificações realizadas, destacam-se:

- Compatibilidade de tipos em expressões e atribuições;
- Verificação de escopo léxico e resolução correta de identificadores;
- Checagem de chamadas de função (número e tipo dos argumentos);
- Validação do uso de operadores pós-fixos apenas em lvalues válidos;
- Consistência de retornos em funções.

Erros semânticos são reportados com mensagens claras, incluindo informações de localização no código-fonte.

2.2 Inferência de tipos

A linguagem SL permite que parâmetros de funções e variáveis sejam declarados sem tipo explícito. Nesses casos, o tipo é inferido a partir do contexto de uso, como atribuições e operações aritméticas.

A inferência é integrada à análise semântica e utiliza as anotações da AST para propagar informações de tipo, garantindo que todos os nós estejam devidamente tipados ao final da análise.

2.3 Interpretador

Foi implementado um interpretador que executa programas SL diretamente a partir da AST anotada. O interpretador segue a semântica operacional da linguagem e suporta:

- Avaliação de expressões aritméticas, lógicas e relacionais;
- Execução de comandos sequenciais e blocos;
- Estruturas de controle (`if-else`, `while`, `for`);
- Chamadas de função com ambiente próprio;
- Manipulação de arranjos e structs;
- Atualização de estado via operadores pós-fixos.

3 Arquitetura do Compilador

O compilador é organizado em módulos bem definidos:

- **Lexer**: reconhecimento de tokens, incluindo palavras reservadas, identificadores, literais, operadores e delimitadores;
- **Parser**: construção da AST com suporte a funções, generics, structs e tipos função;
- **AST**: estrutura parametrizada por anotações para tipos e spans;
- **Semantic**: verificação de tipos, escopo e inferência;
- **Interpreter**: execução de programas SL;
- **PrettyPrinter**: formatação de código com round-trip garantido.

4 Resultados e Discussão

4.1 Funcionalidades Implementadas

Ao final da Entrega 2, o compilador apresenta as seguintes funcionalidades completas:

- Análise léxica e sintática completas;
- Análise semântica com verificação de tipos e escopo;

- Inferência de tipos para parâmetros sem anotação;
- Interpretador funcional para execução de programas SL;
- AST anotada e extensível;
- Pretty printer com round-trip;
- Suporte a operadores pós-fixos.

4.2 Testes

A suíte de testes foi expandida para incluir casos semânticos e de execução, abrangendo:

- Programas semanticamente válidos e inválidos;
- Inferência correta de tipos;
- Execução de programas completos via interpretador;
- Testes de round-trip do pretty printer.

4.3 Limitações

Apesar dos avanços, o compilador ainda não realiza otimizações nem geração de código alvo. Além disso, a inferência de tipos não implementa um sistema completo de unificação (ex: Hindley-Milner), sendo restrita aos casos previstos na linguagem SL.

5 Conclusão

A Entrega 2 consolidou o compilador da linguagem SL como uma ferramenta semanticamente correta e executável. A inclusão da análise semântica, inferência de tipos e interpretador permitiu aplicar conceitos fundamentais de compiladores de forma integrada e prática.

A arquitetura modular adotada facilita a evolução do projeto, permitindo que futuras etapas incluam otimizações e geração de código, sem necessidade de grandes refatorações.

Referências

[1] Hutton, G. (2016). *Programming in Haskell*. Cambridge University Press.

[2] Material didático da disciplina BCC328 – Construção de Compiladores I.

Declaração de uso de IAGen (CONPEP/UFOP)

Esta declaração atende às diretrizes de transparência, autoria, integridade e proteção de dados estabelecidas na Resolução CONPEP nº 144 (UFOP).

Tipo de solicitação:
(i) Revisão e ajuste de trechos do texto do relatório, preservando estrutura e comandos

LaTeX;
(ii) organização/reestruturação de seções e padronização conforme template;
(iii) assistência à programação para refatoração de código haskell do projeto;

Plataforma utilizada:
ChatGPT (modelo GPT-5.2 Thinking) — apoio ao texto e à formatação em LaTeX;
Claude Sonnet 4.5 (Thinking) (via IDE) — apoio à refatoração em Haskell;

Data e horário aproximados da utilização:
Entre 23/01/2026 e 09/02/2026 (aprox.) — refatoração do código e revisão textual e apoio em LaTeX.

Comandos (prompts) usados (resumo):
(1) Solicitar correções pontuais e melhorias de clareza no texto, mantendo LaTeX e sem criar resultados experimentais;
(2) Solicitar reorganização de estrutura e padronização segundo o template;
(3) Solicitar refatoração de módulos Haskell com boas práticas;

Importante:

Durante a preparação deste documento, nós, **Thalles Filipe** e **Julia Gonzaga**, estudantes de **graduação** do curso de **Ciência da Computação** na **UFOP**, declaramos o uso de IAGen **ChatGPT (modelo GPT-5.2 Thinking)** exclusivamente para **finalidades assistivas** (revisão de texto, organização de estrutura e apoio à formatação em LaTeX), sem geração de resultados experimentais. Esse uso foi realizado em conformidade com os **princípios éticos e diretrizes para uso de IAGen** previstos na **Resolução CONPEP nº 144 (UFOP)**, incluindo transparência, autoria, integridade da informação, proteção de dados e preservação da agência humana. Após o uso desta ferramenta, **revisamos e editamos criticamente** o conteúdo, assumindo total responsabilidade pela integridade, originalidade e correção do texto.