Relatório grupo lfa06
BDL e ABDL

André Patacas (93357), Gil Teixeira (88194)

May 2020

1 Resumo

O objetivo deste documento é demonstrar a utilização dos compiladores desenvolvidos e explicitar alguns detalhes de implementação que consideramos relevantes.

Inclui também um breve resumo de como interpretar (compreender) o código das linguagens criadas (instruções sobre como criar código, syntax, etc. estão disponíveis nos readme das linguagens) Os readme estão disponíveis seguindo os links README.md e READMEABDL.md ou na raiz do projeto no code.ua

2 Introdução

Para o projeto foram criados 2 compiladores, um para a linguagem bdl (Board Description Language), que é utilizada para modelar um jogo de tabuleiro, e outro para a linguagem abdl (Auto BDL), que permite criar jogadas sobre o jogo gerado na primeira. A primeira linguagem é utilizada como linguagem auxiliar, já que a sua complexidade está relacionada com o código de suporte na linguagem destino e não com a geração de código em si. A segunda linguagem, por outro lado, conta com a maioria das estruturas de execução de código e operadores comuns como if's, for's, while's, definição de funções, variáveis, etc. e é utilizada como linguagem principal do projeto.

3 Descrição Resumida

3.1 BDL

Na bdl existem 5 secções principais de código:

1. `rules`, nesta secção são descritas as regras mais gerais do código, quem é o primeiro jogador, altura e largura do tabuleiro, qual o padrão de coloração do tabuleiro, e como devem alternar os jogadores. Existe apenas 1 coloração e 1 regra de mudança de jogador disponíveis por default, mas

novas regras podem ser criadas com injeção de código javascript com a keyword explicit.

2. `pieces`, nesta secção são definidas as peças que existem no tabuleiro, e quais os seus movimentos (sentido, comprimento, possibilidade ou não de matar peças inimigas...)

3. `initial_piece_position`, nesta secção são descritas as posições iniciais das peças, bem como o seu dono e se estas aparecem espelhadas verticalmente no tabuleiro, ou seja, cada jogador tem uma réplica da peça.

4. `invariants`, nesta secção são descritos invariantes de jogo, ou seja, regras que nenhuma jogada pode quebrar. São utilizados, por exemplo, na descrição de um jogo de xadrez para explicitar que o "rei" não pode fazer jogadas que permitam o adversário capturá-lo no turno seguinte.

5. `finish`, nesta secção é descrita a regra de fim de jogo. Por default, apenas é disponibilizada a regra que termina o jogo quando não há mais movimentos possíveis. Podem, no entanto, ser adicionadas novas regras através da keyword explicit.

3.2 ABDL

A abdl, por se tratar de uma linguagem de programação dita C-like, é fácil de compreender. A maioria das estruturas comuns da programação estão disponíveis, faltando apenas ressaltar alguns detalhes.

- Variáveis podem ser sobrescritas em inner scopes resolvendo sempre para o significado mais "próximo", tal como em C.

- For loops são na realidade foreach loops sobre uma lista de inteiros, tal como os for loops em python. Alterar a variável de controlo num for loop não a irá alterar na iteração seguinte - para esse efeito deve ser utilizado um while loop.

4 Compilação

Os 2 compiladores estão disponíveis na pasta StandaloneCompiler/, tal como varios exemplos de código nas 2 linguagens. A abdl, por ser algo como um plugin para a bdl, só pode ser compilada sobre um diretório que contenha código gerado pelo compilador da bdl.
Os 2 compiladores têm utilização semelhante:

```
java -jar BDLCompiler.jar [source_file_name] [destination_directory_name]
java -jar ABDLCompiler.jar [source_file_name] [destination_directory_name]
```

5 Execução

O código gerado pelos compiladores é maioritariamente javascript (ES2020) e como tal deve ser corrido através de um servidor. Para o efeito, recomendamos a utilização da extensão do vscode Live Server. Compreendemos a dificuldade que pode surgir e por isso foi criado um curto video em que é compilado e executado um programa em bdl. Pode ser consultado no link https://youtu.be/HfxX0UxSnII

6 Detalhes de implementação

Existem alguns detalhes de implementação que consideramos relevantes e gostaríamos de ressaltar nesta secção.

- Foi utilizado a IDE IntelliJ para o desenvolvimento do projeto em java, recomendamos por isso que o código seja analisado nesse ambiente. Apesar de não ser impossível analisar o código em editores de texto mais simples, existem várias pastas que podem parecer organizadas de forma errada se vistas da perspetiva de um navegador de ficheiros comum. Acrescenta-se ainda que os compiladores foram desenvolvidos de forma separada no tempo e por isso estão em pastas e projetos do IntelliJ diferentes.
- Keyword explicit na bdl que funciona como algo equivalente a actions no antlr ou `__asm` no C. A gramática está desenhada para aceitar matching {}, por default se o número de '{' for igual ao de '}', o input será processado de forma transparente ao user sem necessidade de lidar com caracteres de escape.
- Apesar de por padrão o js permitir redefinição de variáveis em scopes internos, na abdl a funcionalidade foi implementada de raiz recorrendo a uma symbol table desenvolvida especificamente para o trabalho a realizar.
- Inferência dos tipos de dados através do contexto implementados com um segundo visitor chamado TypeInfer que funciona como um interpretador e é chamado múltiplas vezes durante a compilação da abdl.
- Utilização de predicados semânticos para verificar a igualdade do texto de tokens em runtime para definição de funções na abdl.
- Criação de uma classe de handle de erros que permite erros com significado para predicados semânticos não matched.

7 Tratamento de Erros Semânticos

A verificação de erros semânticos para a bdl é praticamente inexistente, já que é uma linguagem simples onde existe pouca semântica associada. Por outro lado, na abdl foi criado um visitor que permite reconhecer inúmeros erros tais como:

- Utilização de variáveis não definidas previamente
- Verificação da coerência de tipos entre operações incluindo tipos de funções chamadas
- Verificação da coerência dos argumentos nas chamadas de funções
- Redefinição indevida de funções
- Inexistência de funções `main` ou `on_move`
- Indexação de variáveis cujo tipo difere de point e utilização de índices não inteiros

8 Trabalho extra

Foi ainda realizado trabalho extra relativo ao projeto.

Foi criado um plugin para o vscode com snippets de código e 2 gramáticas em tmLanguage, para coloração do código desenvolvido em ambas as linguagens. O plugin está disponível na loja pesquisando por bdl ou seguindo este link.

Foram também criadas classes que implementam testes automatizados da aplicação para a verificação de erros semânticos e geração de código da abdl.

Finalmente, foram criados exemplos de código com grau de complexidade elevado de modo a mostrar a capacidade das linguagens desenvolvidas, contando com a programação de um bot que joga uma adaptação de damas chinesas para tabuleiro quadrado.

9 Contribuição dos autores

Os dois autores contribuíram igualmente para o projeto.

- André Patacas 93357 (50%)
- Gil Teixeira 88194 (50%)