Ferramenta educacional para auxiliar no ensino do modelo fork-join

Lucas da Silva Marcos[1], **Rodolfo Adamshuk Silva**[1]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Dois Vizinhos, Paraná

lucasmarcos@alunos.utfpr.edu.br, rodolfoa@utfpr.edu.br

Abstract.

Resumo. *A importância da programação concorrente no campo da ciência da computação é inegável. No entanto, a complexidade inerente ao ensino desta disciplina em ambientes acadêmicos é um desafio significativo. Este artigo propõe um compilador como ferramenta educacional destinada a facilitar o ensino do modelo de programação fork-join, um paradigma utilizado na programação concorrente.*

1. Introdução

Um programa concorrente é um programa que contém dois ou mais processos que trabalham juntos para realizar uma tarefa [and]. A programação concorrente consiste em programar em uma linguagem que permite indicar explicitamente como diferentes partes de um cálculo podem ser executadas concorrentemente por diferentes processadores [qui]. Algumas das linguagens de programação mais populares possuem suporte ao uso de programação concorrente, como Java, Python, C, Go, Elixir, entre outras. Para criar programas concorrentes de qualidade, o programador deve ter conhecimento de conceitos básicos, como criação e finalização de processos, distribuição de carga de trabalho, comunicação e sincronização de processos, inerentes a esse tipo de programação.

A utilização de conceitos de programação pode introduzir uma nova camada de complexidade aos programadores, principalmente aos estudantes que ainda não possuem muita experiência em programação. A compreensão desses conceitos é essencial para o desenvolvimento de sistemas robustos e escaláveis.

O ensino de programação concorrente enfrenta um desafio significativo devido à natureza complexa e abstrata dos conceitos envolvidos. A falta de experiência dos estudantes e a escassez de ferramentas de suporte ao ensino podem dificultar a aprendizagem dos conceitos de programação concorrente.

O modelo fork-join, introduzido por Conway [Conway 1963] e Dennis e Van Horn [Dennis and Van Horn 1966], é um conceito simples para a criação de processos. Essa foi a primeira notação de linguagem para especificação de concorrência. Por ser um modelo simples de ser implementado, é utilizado por alguns professores para o ensino de programação concorrente. Durante o ensino desse modelo, os estudantes realizam a especificação de grafo de dependências em um pseudo-código. Mesmo utilizando um modelo mais simples de programação concorrente, nota-se que os estudantes ainda possuem dificuldades para utilizar a sintaxe do modelo.

O objetivo desse trabalho é a criação de um avaliador léxico e sintático para ser utilizado no apoio do ensino de programação concorrente no contexto de uma disciplina de programação concorrente de uma universidade federal. O avaliador será desenvolvido utilizando conceitos de compiladores para a verificação da sintaxe dos comandos de fork-join, onde o compilador irá realizar a análise léxica e sintática dos códigos. Por meio dessa avaliação, será possível encontrar enganos no uso da sintaxe do modelo.

Algumas abordagens na literatura apresentam o uso de ferramentas educacionais para o apoio ao ensino de programação para estudantes universitários. Por exemplo, ensino da linguagem Logo [Raiol et al. 2015], programação concorrente [bra] e blocos visuais [Cardoso and Faria 2019].

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta o referencial teórico do trabalho, a Seção 3 apresenta a proposta do avaliador sintático e a Seção 4 apresenta as atividades a serem desenvolvidas durante o projeto.

2. Trabalhos relacionados

A comunicação entre processos, que pode ocorrer por exemplo por memória compartilhada ou passagem de mensagem. problema x tarefa x processo.

2.3. Trabalhos relacionados [Raiol et al. 2015] apresentam uma ferramenta para o ensino de programação Logo para estudantes do 1 ano do curso de sistemas de informação. Essa ferramenta ajuda a desenvolverem ou ampliarem suas habilidades com a programação. [Cardoso and Faria 2019] apresenta a ferramenta Scratch no ensino superior. A tecnologia é um forte aliada a educação. Ótimo agregador de benefícios a educação.

Nessa seção estão descritos os tópicos mais importantes e os trabalhos relacionados para a fundamentação teórica da ferramenta proposta.

2.1. Programação concorrente

[Van Steen and Tanenbaum 2017] e [Coulouris et al. 2001].

É uma maneira de organizar um programa em vários fluxos de execução. A pricilpal dificuldade em programação concorrente é a comunicação e coordenação entre processos. As duas maneiras principais que essa comunicação ocorre é por memoria compartilhada e passagem de mensagem.

Na memória compartilhada os processos concorrentes compartilham um espaço de memória comum, permitindo a comunicação direta entre eles por meio da leitura e escrita em variáveis compartilhadas.

E na passagem de mensagem, os processos concorrentes se comunicam trocando mensagens entre si, sem compartilhar diretamente memória. As mensagens podem ser enviadas e recebidas através de canais de comunicação específicos.

Também encontramos em programação concorrente os conceitos de problema, tarfea e processo. Problema refere-se a uma questão ou desafio a ser resolvido por um sistema computacional. Tarefa é uma unidade de trabalho dentro de um programa ou sistema, geralmente relacionada à resolução de uma parte específica de um problema maior. Processo é uma instância em execução de um programa em um sistema computacional. Cada processo tem seu próprio espaço de memória e fluxo de execução, podendo realizar múltiplas tarefas de forma concorrente.

2.2. Fork-join

[Conway 1963] e [Dennis and Van Horn 1966].

Um exemplo de código fork-join.

2.3. Compiladores

Compiladores são programas de computador que traduzem de uma linguagem para outra [Louden 2004].

As fases de um compilador incluem:

Análise Léxica. Análise Sintática. Análise Semântica. Geração de Código Intermediário. Otimização de Código. Geração de Código Executável.

A análise léxica é a primeira fase do processo de compilação, onde o código-fonte é lido e convertido em tokens. Esses tokens representam os componentes básicos da linguagem, como palavras-chave, identificadores, operadores e símbolos especiais.

A análise sintática é a segunda fase do processo de compilação, onde os tokens gerados pela análise léxica são organizados em uma estrutura hierárquica que representa a sintaxe do programa. Esta fase verifica se o código-fonte está gramaticalmente correto de acordo com as regras da linguagem de programação.

2.4. Recursos educacionais para ensino de programação

[Raiol et al. 2015] e [Cardoso and Faria 2019].

3. Proposta

Com a necessidade de aprender e pôr em prática os conceitos de programação concorrente, o modelo de fork-join é utilizado como primeira ferramenta de especificação de programas concorrentes. O modelo fork-join apresenta uma sintaxe para a criação de processos e união de fluxos.

A unidade curricular de programação concorrente para a qual o avaliador sintático será definido utiliza uma sintaxe específica para a utilização da função de criação e união de fluxos de execução.

A instrução Fork é utilizada para representar a bifurcação da execução. Ao usar a instrução Fork, é criado um novo fluxo de execução. A sintaxe da instrução Fork é a seguinte: FORK <rótulo> onde <rótulo> representa o nome de uma função que será executada no novo fluxo de execução.

A instrução Join agrupa duas execuções concorrentes em uma. . . .

JOIN var-ctrl , <rot1> , <rot2>;

O fork-join é utilizado no contexto da unidade curricular de programação concorrente para especificar grafos de fluxo de controle. A Figura X apresenta um grafo de fluxo de controle, onde cada nó representa uma tarefa e cada aresta representa uma dependência entre as tarefas. O Código X apresenta a especificação do grafo da Figura X utilizando a sintaxe do modelo fork-join.

O objetivo deste trabalho é a criação de uma ferramenta que irá fazer a avaliação léxica e sintática de códigos escritos usando o modelo fork-join. O propósito desta ferramenta é auxiliar os estudantes na avaliação de seus códigos desenvolvidos com o modelo fork-join. A ferramenta irá receber um código e realizar indicações de problemas na estrutura léxica e sintática do código. Essas indicações serão apresentadas por meio de uma interface gráfica e mensagens de erro.

O avaliador consistirá em dois módulos: 1) avaliador léxico e 2) avaliador sintático. A Figura X apresenta uma visão geral da ferramenta.

Figura Entrada (código Fork-join) Atividades ("XXX", Análise léxica, análise sintática, "compilação", gerador de erros)

Análise léxica: Nessa fase será utilizada a entrada com o código do usuário. A primeira etapa consiste na leitura do arquivo. . . . . .

Análise sintática:

Como resultado, muitos alunos enfrentam dificuldades ao desenvolver e depurar programas baseados neste paradigma. O desenvolvimento de um compilador e plataforma com o foco no uso em sala de aula. Será especificado uma gramática formal da linguagem de alto nível utilizada. A linguagem possuirá os primitivos Fork e Join como modelo de concorrência. Será construído um analisador léxico, reportando aos alunos erros nos programas. Será construído um analisador sintático, que modelará a estrutura do programa e reportará erros de construção. Como saída esperada teremos, além das mensagens de erro ou de sucesso, um grafo de tarefas e dependências, além de métricas do programa especificado, como o caminho crítico e o grau máximo de concorrência. Também será construído uma plataforma web, como um editor de textos integrado e renderização do grafo e métricas. Um exemplo de plataforma sendo o Compiler Explorer https://godbolt.org.

desafios adicionais para os estudantes, especialmente em relação à decomposição adequada de tarefas e à gestão eficiente de recursos compartilhados.

este trabalho tem por objetivo desenvolver uma ferramenta de ensino que auxilie em tais objetivos.

O objetivo principal deste trabalho é desenvolver um compilador que

O compilador fornecerá recursos de análise léxica e sintática para identificar erros no código.

A motivação para o desenvolvimento de uma ferramenta educacional que suporte o ensino do modelo fork-join em uma linguagem de alto nível surge da necessidade de tornar os conceitos de concorrência mais acessíveis e tangíveis para os estudantes.

Além disso, a ferramenta educacional incluirá recursos para visualização da decomposição de tarefas em forma de grafo, juntamente com métricas associadas para avaliar o desempenho e a eficiência dos programas desenvolvidos.

Ao fornecer uma linguagem de programação abstrata e familiar, enriquecida com operações específicas do modelo fork-join, espera-se reduzir a curva de dificuldade no aprendizado e permitir que os alunos se concentrem nos conceitos fundamentais, em vez de se preocuparem com detalhes de implementação.

4. Metodologia

Será um pesquisa aplicada e exploratória.

Aplicaremos o protótipo em sala de aula, coletaremos dados e feedback dos alunos. O feedback será uma pesquisa qualitativa com as opiniões dos alunos sobre o uso do sistema em um formulário online.

O cronograma das atividades proposta está descrito abaixo.

4.1. Atividades

- **Atividade 1**: atividade 1;
- **Atividade 2**: atividade 2.

	Julho	Agosto	Setembro	Outubro	Novembro	Dezembro
Atividade 1	■					
Atividade 2		■				
Atividade 3			■			

5. Resultados esperados

Espera-se que a ferramenta proposta contribua significativamente para o aprimoramento do ensino de programação concorrente, fornecendo um recurso para o aprendizado dos alunos e auxiliando os professores no processo de ensino.

Referências

Cardoso, L. R. and Faria, D. d. S. E. (2019). O uso do scratch como ferramenta de auxílio no ensino superior. *Anais do Seminário Científico do UNIFACIG*, (5).

Conway, M. E. (1963). A multiprocessor system design. In *Proceedings of the November 12-14, 1963, fall joint computer conference*, pages 139–146.

Coulouris, G., Dollimore, J., and Kindberg, T. (2001). *Sistemas distribuidos*, volume 6. Addison Wesley Madrid.

Dennis, J. B. and Van Horn, E. C. (1966). Programming semantics for multiprogrammed computations. *Communications of the ACM*, 9(3):143–155.

Louden, K. (2004). *Compiladores - Princípios e Práticas*. Pioneira Thomson Learning.

Raiol, A. A., Sarges, J., Souza, A., Silva, S., and BEZERRA, F. d. L. (2015). Resgatando a linguagem de programaçao logo: Uma experiência com calouros no ensino superior. WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO.

Van Steen, M. and Tanenbaum, A. S. (2017). *Distributed systems*. Maarten van Steen Leiden, The Netherlands.