Ferramenta educacional de suporte ao 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

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 [Andrews 2001]. 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 [Quinn 2003]. 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 concorrente 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. Além disso, 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 e 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 pseudocódigo. 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 uma ferramenta de suporte ao ensino de programação concorrente no contexto de uma unidade curricular de programação concorrente de uma universidade federal. A ferramenta será desenvolvida utilizando conceitos de compiladores para a verificação da sintaxe dos comandos de fork-join, no qual o compilador realizará a análise léxica e sintática dos códigos. Espera-se que o avaliador auxilie os estudantes a encontrar possíveis enganos no uso da sintaxe do modelo fork-join.

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 [Braga et al. 2015] e blocos visuais [Cardoso e 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. Referencial Teórico

2.1. Fork-Join

O modelo **fork-join** é um paradigma de programação concorrente que permite a decomposição de um problema em tarefas independentes que podem ser executadas em paralelo e, posteriormente, reunidas em um único resultado. Esse modelo é particularmente útil para tarefas de granularidade variável e tem ampla aplicação em computação paralela e distribuída [Conway 1963, Dennis e Van Horn 1966].

A Figura 1 é uma representação do funcionamento do modelo fork-join. A instrução FORK é utilizada para representar a bifurcação da execução, ou seja, cria um fluxo de execução. Dessa forma, ela possui dois sucessores e funciona como uma instrução de desvio condicional. Na figura, tem-se um FORK na linha 100 e as instruções das linhas 101 e 200 serão executadas em seguida. As linhas de 101 até 199 correspondem a Tarefa A e as linhas de 200 até 299 correspondem a Tarefa B. A execução de uma instrução FORK ativa outro processador, caso esteja disponível.

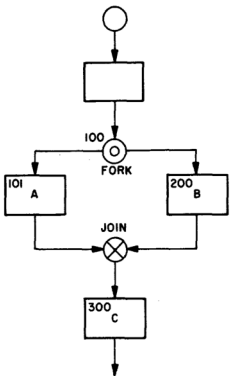


Figura 1. fork-join. Fonte: Adaptado de Conway [Conway 1963].

2.2. Compiladores

Compiladores são programas de computador que traduzem código-fonte de uma linguagem de programação para outra, geralmente de uma linguagem de alto nível para código

de máquina [Louden 2004]. A sintaxe de uma linguagem de programação é normalmente dada pelas regras gramaticais de uma gramática livre de contexto, que é um tipo de gramática formal utilizada para definir as possíveis estruturas de sentenças na linguagem. Uma gramática livre de contexto consiste em um conjunto de regras de produção que especificam como os símbolos não-terminais podem ser substituídos por combinações de símbolos terminais e não-terminais, independentemente do contexto em que aparecem [Louden 2004].

O processo de compilação é divido em seis etapas [Louden 2004]. Primeiramente, a **análise léxica** converte o código-fonte em tokens, que são os elementos básicos da linguagem de programação. Em seguida, a **análise sintática** organiza os tokens em uma estrutura hierárquica, verificando a correção gramatical do código-fonte. Na terceira etapa, a **análise semântica** verifica o significado e a semântica do código-fonte, garantindo a correção dos tipos de dados e escopo das variáveis. Após isso, a **geração de código intermediário** cria uma representação intermediária do programa original. A **otimização de código** melhora a eficiência e desempenho do código intermediário. E a **geração de código executável** traduz o código intermediário otimizado em código executável específico para a plataforma de destino.

2.3. Recursos Educacionais para Ensino de Programação

Recursos educacionais adequados, como ferramentas de desenvolvimento e exercícios práticos são essenciais para auxiliar os alunos no desenvolvimento de habilidades sólidas na área de programação.

Raiol et al. [Raiol et al. 2015] apresentam uma ferramenta para o ensino de programação Logo para estudantes do primeiro ano do curso de sistemas de informação, que ajuda a desenvolver e ampliar suas habilidades de programação. [Cardoso e Faria 2019] apresentam a ferramenta Scratch no ensino superior, destacando a tecnologia como uma aliada forte na educação e agregadora de benefícios.

3. Proposta

A unidade curricular de programação concorrente para a qual a ferramenta será criada utiliza uma sintaxe específica para a utilização da função de criação e união de fluxos de execução. A sintaxe da instrução FORK (que realiza a criação de fluxos) é a seguinte:

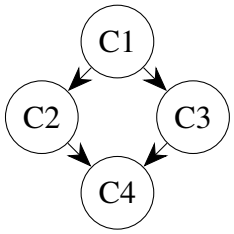
```
FORK <rótulo>;
```

onde <rótulo> representa o nome de uma função que será executada no novo fluxo de execução.

A sintaxe da instrução JOIN (que realiza a união de fluxos) é:

```
JOIN <variável-de-controle>, <rótulo-1>, <rótulo-2>;
```

Como o JOIN pode controlar vários fluxos de processos, a instrução JOIN utiliza uma variável de controle <variável-de-controle> para controlar quantos fluxos ainda estão ativos. Quando um fluxo executa o JOIN, ele verifica se ele é o último. Se for, ele executa o <rótulo-1>. Caso contrário, ele irá executar o <rótulo-2.>. Para finalizar um processo, utilizamos o rótulo QUIT.



(a) Grafo

```
VAR_J = 2;
C1;
FORK ROT_C3;
C2;
JOIN VAR_J, ROT_C4, QUIT;

ROT_C3:
 C3;
 JOIN VAR_J, ROT_C4, QUIT;

ROT_C4:
  C4;
```

(b) Código

Figura 2. fork-join

O fork-join é utilizado no contexto da unidade curricular de programação concorrente para especificar grafos de fluxo de controle. A Figura 2(a) apresenta um grafo de fluxo de controle, onde cada nó representa uma tarefa e cada aresta representa uma dependência entre as tarefas. A Figura 2(b) apresenta a especificação do grafo utilizando a sintaxe do modelo fork-join.

O propósito desta ferramenta é auxiliar os estudantes na avaliação de seus códigos desenvolvidos com o modelo fork-join. A ferramenta receberá um código e realizará indicações de problemas na estrutura léxica e sintática do código. Essas indicações serão apresentadas por meio de uma interface gráfica e mensagens de erro.

A ferramenta consistirá em três módulos: 1) analisador léxico, 2) analisador sintático e 3) geração de grafo de dependências. A Figura 3 apresenta uma visão geral da ferramenta.

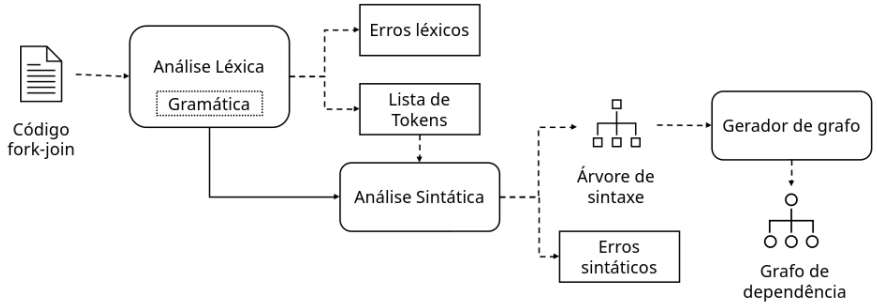


Figura 3. Visão geral do projeto

A análise léxica é a primeira fase de um processo de compilação e sua função é fazer a leitura do programa fonte, caractere a caractere, agrupar os caracteres em lexemas e produzir uma sequência de símbolos léxicos conhecidos como tokens. A sequência de tokens é enviada para ser processada pela análise sintática que é a próxima fase do processo de compilação.

A análise sintática determina se o programa de entrada representado pelo fluxo de tokens possui as sentenças válidas para a linguagem de programação. A análise sintática é a segunda etapa do processo de compilação, e na maioria dos casos, utiliza gramáticas livres de contexto para especificar a sintaxe de uma linguagem de programação.

A gramática formal que será definida é fundamental no desenvolvimento dos analisadores léxico e sintático. Esta gramática especificará as regras e estruturas sintáticas

que o código deve seguir, servindo como referência para a validação dos programas escritos no modelo fork-join. Durante a análise léxica do código-fonte, caso algum token identificado não corresponda aos padrões estabelecidos pela gramática da linguagem, ele será classificado como um erro léxico. Quando a estrutura de uma sentença não estiver em conformidade com as regras estabelecidas pela gramática definida ocorrerão erros de sintaxe. Esses erros serão apresentados com mensagens apropriadas, indicando a natureza e a localização dos enganos.

O gerador de grafo, utilizando a árvore de sintaxe gerada pelo analisador sintático, cria uma representação gráfica do código. Este grafo ilustra as relações de fork-join entre diferentes segmentos do código, facilitando a visualização das dependências e paralelismos presentes no código analisado. A representação gráfica ajudará os estudantes a compreenderem a estrutura e o fluxo do programa concorrente.

4. Atividades e Cronograma

Para alcançar o objetivo desse trabalho, será conduzida uma metodologia composta por seis atividades. O cronograma das atividades propostas está descrito na Tabela 1.

Criação da gramática do modelo fork-join: Nessa atividade será desenvolvida a gramática baseada na estrutura do modelo. Para isso, será feita a definição da sintaxe e das regras gramaticais necessárias para representar o modelo fork-join. A gramática servirá como base para o desenvolvimento dos analisadores léxico e sintático.

Criação do analisador léxico: Com base na gramática desenvolvida na atividade anterior será desenvolvido um analisador léxico que irá realizar a análise dos códigos. O analisador percorrerá o código-fonte, identificando e classificando os elementos léxicos (tokens) como palavras-chave, identificadores, operadores, etc. Como resultado, será gerada uma lista de tokens para cada código. Caso haja enganos no código, o analisador gerará uma lista de problemas léxicos indicando onde ocorreram os erros.

Criação do analisador sintático: Com base na lista de tokens gerada na atividade anterior, será realizada a análise sintática do código. O analisador sintático verificará se a sequência de tokens segue as regras da gramática definida. Durante esse processo será construída uma árvore de sintaxe abstrata (AST) que representará a estrutura hierárquica do código. Caso haja algum problema de sintaxe, o analisador gerará uma lista de erros sintáticos indicando os pontos de falha no código.

Criação do gerador de grafo: Com base na árvore de sintaxe gerada na atividade anterior, o gerador de grafo criará uma representação gráfica do código. Este grafo ilustrará as relações de fork-join entre diferentes segmentos do código, mostrando visualmente a estrutura paralela do programa. A geração do grafo facilitará a compreensão das dependências e paralelismos presentes no código analisado.

Teste da ferramenta: Nesta fase serão realizados testes das funcionalidades e de aceitação do sistema. Serão utilizados exemplos de códigos de programas concorrentes para verificar se as saídas geradas pelo sistema são corretas e úteis. Serão coletados *feedbacks* de estudantes de programação concorrente para avaliar a eficácia da ferramenta como auxílio educacional. Ajustes e melhorias serão feitos com base nos resultados dos testes.

Escrita do TCC 2: Nesta atividade será realizada a escrita do Trabalho de Con-

clusão de Curso 2. O documento incluirá a descrição detalhada de todas as etapas do desenvolvimento da ferramenta, os resultados obtidos nos testes, a análise crítica das funcionalidades e a contribuição do trabalho para o ensino de programação concorrente.

	Agosto	Setembro	Outubro	Novembro	Dezembro
Gramática	■				
Análise léxica		■			
Análise sintática			■		
Gerador				■	
Teste					■
Escrita	■	■	■	■	■

Tabela 1. Cronograma de atividades.

5. Resultados esperados

O resultado esperado desse trabalho é a disponibilização de uma ferramenta web de suporte à especificação de programas usando o modelo fork-join.

Referências

Andrews, G. (2001). *Foundations of Multithreaded, Parallel, and Distributed Programming*. Addison-Wesley.

Braga, S. A., Bezerra, H. C., e Garcia, F. P. (2015). "crianças brincando": Uma ferramenta para o auxílio na aprendizagem de programação concorrente.

Cardoso, L. R. e 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.

Dennis, J. B. e 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.

Quinn, M. J. (2003). *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Education Group.

Raiol, A. A., Sarges, J., Souza, A., Silva, S., e Bezerra, F. d. L. (2015). Resgatando a linguagem de programaçao logo: Uma experiência com calouros no ensino superior. Workshop sobre educaço em computaçaõ.