

Ferramenta educacional de suporte ao ensino do modelo fork-join

Lucas da Silva Marcos¹, Rodolfo Adamshuk Silva¹

¹Universidade 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 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 [Braga et al. 2015] 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. 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] e [Dennis and Van Horn 1966].

As instruções de fork e join em fluxogramas possuem seus equivalentes nas instruções FORK e JOIN, que são incorporadas ao conjunto de instruções do sistema. A instrução FORK é essencialmente uma instrução com dois sucessores e funciona de maneira semelhante a uma instrução de desvio condicional. No entanto, se a localização 100 contiver uma instrução FORK 200, as instruções nas localizações 200 e 101 serão subsequentemente executadas. A execução de uma instrução FORK ativa outro processador, caso esteja disponível. É importante notar que a instrução FORK possui uma propriedade associativa; N caminhos paralelos podem ser especificados de forma igualmente eficaz por muitos arranjos possíveis de N-1 forks.

A instrução JOIN, que é, na prática, o inverso da FORK, tem uma função adicional crucial: ela aguarda. No Diagrama 1, a execução da caixa C não deve começar até que as caixas A e B sejam concluídas. Suponha que o código para a caixa A vá das localizações 101 a 105, o código para a caixa B vá das localizações 200 a 219, e o código para a caixa C comece na localização 300. Após a execução da FORK na localização 100, dois processadores são ativados: um executa as instruções de 101 a 105, e o outro executa as instruções de 200 a 219. O processador que terminar primeiro, por exemplo, na localização 105, deve ser liberado; o que terminar por último então simplesmente fará um desvio para a localização 300.

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]. O processo de compilação é dividido em seis etapas. A análise

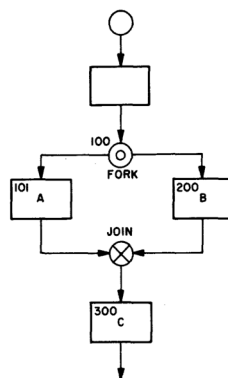


Figura 1. fork-join

léxica converte o código-fonte em tokens, que são os elementos básicos da linguagem de programação. A análise sintática organiza os tokens em uma estrutura hierárquica, verificando a correção gramatical do código-fonte. A análise semântica verifica o significado e a semântica do código-fonte, garantindo a correção dos tipos de dados e escopo das variáveis. 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

Para o ensino de programação concorrente, é fundamental fornecer materiais que abordem conceitos específicos, como o modelo fork-join, e exemplos práticos de implementação de sistemas concorrentes. Recursos educacionais adequados, como ferramentas de desenvolvimento e exercícios práticos, são essenciais para auxiliar os alunos no desenvolvimento de habilidades sólidas nessa área [Raiol et al. 2015], [Cardoso and Faria 2019].

[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 and Faria 2019] apresentam a ferramenta Scratch no ensino superior, destacando a tecnologia como uma aliada forte na educação e agregadora de benefícios.

Esses trabalhos relacionados são fundamentais para a fundamentação teórica da ferramenta proposta, pois fornecem insights sobre abordagens eficazes de ensino e ferramentas educacionais no campo da programação concorrente.

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 compilador 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. Cada uma das execuções deve requisitar para ser agrupada com a outra. Já que as computações podem ter tempos de execução diferentes, uma pode executar o JOIN antes da outra e sua execução é terminada. Como o JOIN pode controlar vários fluxos de processos, a instrução JOIN utiliza uma variável de controle para controlar quantos fluxos ainda estão ativos. A sintaxe dessa instrução é:

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

Considerando uma variável de controle <variável-de-controle>, e dois fluxos concorrentes <rótulo-1> e <rótulo-2>. Para finalizar um processo, utilizamos o rótulo QUIT.

O fork-join é utilizado no contexto da unidade curricular de programação concorrente para especificar grafos de fluxo de controle. A Figura 2 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 apresenta a especificação do grafo da Figura 2 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 três módulos: 1) analisador léxico, 2) analisador sintático e 3) geração de grafo de dependências. A figura 3 apresenta uma visão geral da ferramenta.

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.

O Analisador sintático tem como tarefa principal determinar se o programa de entrada representado pelo fluxo de tokens possui as sentenças válidas para a linguagem de programação. A análise sintática e 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.

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

```

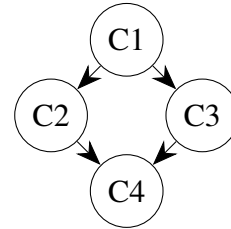
VAR_J = 2;
C1;
FORK ROT_C3;
C2;
JOIN VAR_J, ROT_C4, QUIT;

ROT_C3:
  C3;
  JOIN VAR_J, ROT_C4, QUIT;

ROT_C4:
  C4;

```

(a) Código



(b) Grafo

Figura 2. fork-join

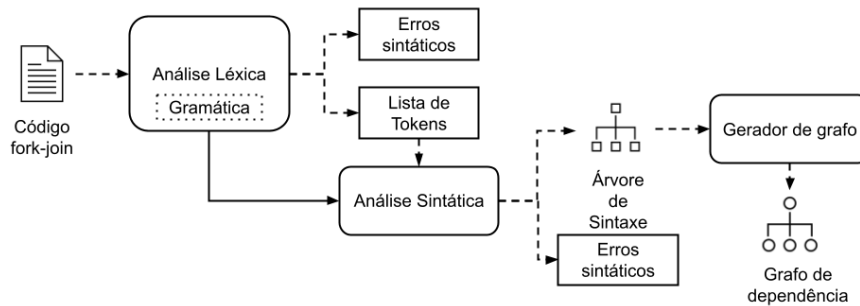


Figura 3. Visão geral do projeto

entre diferentes segmentos do código, facilitando a visualização das dependências e paralelismos presentes no código analisado. A representação gráfica ajuda os estudantes a compreenderem a estrutura e o fluxo do programa concorrente.

4. Atividades e Cronograma

Para alcançar o objetivo do presente 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 Conclusão 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

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., and Garcia, F. P. (2015). “crianças brincando”: Uma ferramenta para o auxílio na aprendizagem de programação concorrente.
- 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.

- 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.
- 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., and Bezerra, F. d. L. (2015). Resgatando a linguagem de programação logo: Uma experiência com calouros no ensino superior. Workshop sobre educação em computação.