Code Umpire: uma abordagem para avaliação automática e unificada de restrições em código-fonte de aprendizes de programação

Kleber Tarcísio Oliveira Santos[1], Galileu Santos de Jesus[1], Jaine da Conceição Santos[1], Alberto Costa Neto[1]

[1]Departamento de Computação – Universidade Federal de Sergipe
Av. Marechal Rondon, Jardim Rosa Elze, São Cristóvão - SE, 49100-000

klebertarcisio@yahoo.com.br, galilasmb@gmail.com,

jainecs@dcomp.ufs.br, alberto@ufs.br

Abstract. *This paper presents an unified approach to specify and check source code constraints supported by a static analyzer named Code Umpire. This tool was developed based on the collection and analysis of 497 problems and more than 10,000 submissions from the programming problems database of the online judge The Huxley. It was observed that student's main difficulty is exercising the creation and use of recursive functions. In addition, this work makes a comparison with other approaches that are also supported by a static analyzer.*

Resumo. *Este artigo apresenta uma abordagem unificada de especificação e checagem de restrições de código-fonte apoiada por um analisador estático chamado Code Umpire. Esta ferramenta foi desenvolvida após uma coleta e análise de 497 problemas e mais de 10.000 submissões de problemas de programação da base de dados do juiz on-line The Huxley. Foi detectada, principalmente, a dificuldade que os alunos têm em exercitar a criação e utilização de funções recursivas. Além disso, este trabalho faz uma comparação com outras abordagens que também são apoiadas por um analisador estático.*

1. Introdução

Os juízes *on-line* surgiram como uma alternativa para acompanhar numerosas turmas no ensino de programação. Utilizando estas ferramentas, os professores podem acompanhar individualmente cada um dos alunos, e estes podem realizar as atividades práticas de programação de qualquer lugar e a qualquer momento, obtendo retorno em tempo real sobre as correções de suas soluções implementadas [Bez et al. 2014, Sun and Bofang Li 2014]. Isso faz aumentar o interesse do aluno e consequentemente melhora sua capacidade de resolver problemas e de programar [de Barros Paes et al. 2013, Wu and Shuangping Chen 2012].

Apesar da existência de diversos sistemas de juiz *on-line* que facilitam o processo de ensino e aprendizagem nas disciplinas de programação, ainda há o que aprimorar. Este trabalho concentra-se na dificuldade que o professor tem de forçar o aluno a resolver os problemas conforme o que foi especificado. Como o público-alvo são alunos que estão aprendendo a programar e por isso estão ainda exercitanto o uso de construções simples de programação, como estrutura condicional, estrutura de repetição, função, função recursiva, declaração de variáveis e outros, este aluno geralmente prefere utilizar conceitos que

já absorveu para aplicar na solução a utilizar as construções especificadas no problema, as quais chamamos de restrições estruturais, para resolvê-lo.

Este artigo contém dados estatísticos que evidenciam que não é atípico o professor solicitar que um problema seja resolvido com uma função recursiva e o aluno apresentar uma solução com alguma construção iterativa (*for*, *while* ou *do while*). Este trabalho buscou concentrar esforços para resolver problemas de situações como essa.

Ao longo dos anos, foram desenvolvidas várias técnicas para enfrentar problemas dessa espécie. De maneira geral, estas técnicas estão agrupadas em um conjunto chamado de Análise Estática [Ala-Mutka 2005]. Esta análise envolve a coleta de informações sobre o código-fonte sem precisar executá-lo. Por meio desta abordagem, é possível realizar diversas checagens, como: verificar erros sintáticos na solução do aluno e dar alguma dica de como corrigi-lo, verificar o grau de similaridade entre dois ou mais algoritmos, verificar se a solução do aluno apresenta um conjunto predefindo de palavras-chaves. Um estudo mais aprofundado dessas e de outras técnicas pode ser encontrado em [Rahman and Nordin 2007].

Este artigo está dividido da seguinte forma: a Seção 2 apresenta uma série de dados que evidencia a necessidade do professor utilizar um analisador estático no ensino de programação. A Seção 3 traz os principais trabalhos encontrados na literatura e principalmente como eles tratam a análise estática. A Seção 4 apresenta os detalhes do funcionamento do Code Umpire. A Seção 5 faz uma comparação entre os trabalhos relacionados e o Code Umpire. Finalmente, a Seção 6 apresenta as conclusões.

2. Avaliação estática no ensino de programação

Esta seção apresenta três conjuntos de dados que evidenciam a necessidade de utilização de um analisador estático no ensino de programação.

The Huxley é uma ferramenta *web* que permite aos alunos submeterem código-fonte em diversas linguagens de programação como respostas a exercícios de uma base com centenas de problemas. Para cada submissão, o aluno recebe *feedback* da correção automática pelo sistema através de análise sintática do código e dos testes de aceitação. Foi pensado para auxiliar o aluno e o professor dentro e fora de sala de aula [de Barros Paes et al. 2013]. Esta ferramenta possui uma classificação para cada problema, chamada de Nível de Dificuldade (ND). Atualmente, o ND de cada problema pode variar de 1 a 5 (iniciante, fácil, médio, difícil e expert). Por conveniência, foi decidido selecionar apenas os problemas com ND de 1 a 3, por se tratarem de problemas mais simples que podem ser resolvidos por alunos que estão aprendendo a programar.

Diante dos 497 problemas selecionados para análise, foi realizada uma leitura do enunciado de todos eles com o objetivo de descobrir quantos exigiam explicitamente algum tipo de restrição na solução. Foi observado que 76 problemas (15,29%) exigiam que o aluno oferecesse uma solução com algum tipo de restrição. Vale ressaltar que este valor poderia ser maior, já que no momento da coleta o The Huxley ainda não suportava a análise estática. Desta forma, os professores e os alunos sabiam que suas soluções não seriam realmente avaliadas estaticamente. A Figura 1 apresenta um mapeamento de todas as restrições encontradas nos 76 problemas.

Apesar de a Figura 1 demonstrar que há uma necessidade de o professor exigir que o aluno implemente sua solução com determinados recursos de programação (laço,

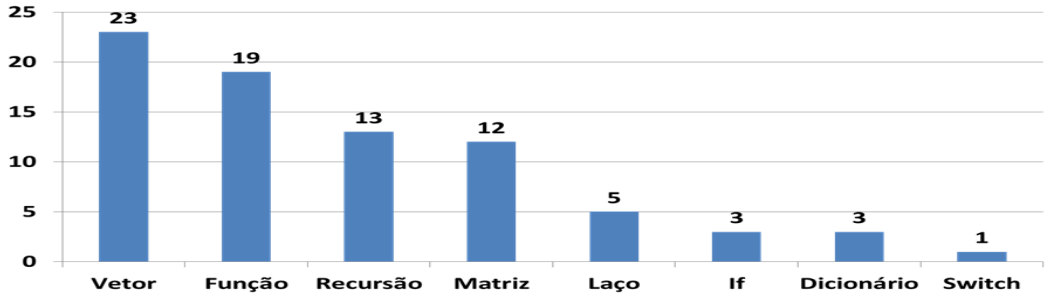

Figura 1. Mapeamento das restrições

função, *if* etc), o desenvolvimento de um analisador estático somente se justifica caso seja observado que os alunos não seguem as restrições impostas pelos professores. Um analisador estático não teria grande utilidade se percebido que os alunos já resolvem com facilidade os problemas da forma como professor solicitou.

Tabela 1. Resumo do atendimento às restrições

Restrição	Submissões	Atenderam à restrição	Percentual
Array/Matriz	3138	3089	98%
Repetição	147	114	78%
Dicionário	101	78	77%
IF	798	564	71%
Função	2972	1523	51%
Função Geral	5926	2463	47%
Recursão	2230	940	42%

O segundo conjunto de dados é apresentado na Tabela 1, que mostra que em diversos casos os alunos não seguem as restrições estruturais solicitadas pelos professores. Inicialmente, os autores deste trabalho perceberam que diversas soluções enviadas pelos alunos não seguiam as restrições estabelecidas pelos professores. Entretanto, é árduo e pouco produtivo que pessoas avaliem estaticamente mais de 10.000 soluções de programação que poderiam ser avaliadas por uma ferramenta. Dessa forma, os dados foram adquiridos após o desenvolvimento da ferramenta Code Umpire, todavia já havia um indicativo do não atendimento às restrições que posteriormente foi comprovado com os dados da Tabela 1.

De acordo com a Tabela 1, a restrição menos atendida pelos alunos foi implementar uma função recursiva com um determinado nome e quantidade de parâmetros. É notável que essa restrição é a mais fácil de não ser atendida porque o problema pode ser resolvido com outras alternativas. O aluno pode utilizar um laço de repetição (*for*, *while* ou *do while*) ao invés da recursão e os testes caixa-preta (testes de aceitação) não identificariam erros. Outra maneira de não seguir a restrição é sequer criar uma função e não praticar a capacidade de modularizar os programas. Um documento com todos os dados detalhados por problema, tipo de restrição e linguagem de programação está disponível no Dropbox[1].

O terceiro conjunto de dados advém de um questionário aplicado a 38 professores com experiência em turmas introdutórias de programação. Houve uma divulgação na lista de discussão da Sociedade Brasileira de Computação, o que permitiu que qualquer pro-

[1]https://www.dropbox.com/s/m1hauw0rr0ae357/dados_the_huxley.pdf?dl=0

fessor interessado participasse. O questionário foi feito por meio da plataforma Google Forms e disponibilizado por *e-mail* para profissionais da área. Através deste questionário, os professores informaram quais restrições estruturais usariam com mais frequência em suas turmas se possuíssem uma ferramenta que permitisse especificar e checar tais restrições. O questionário completo com as respostas está disponível no Dropbox[2]. Das restrições sondadas, as que mais se destacaram foram:

- Criação de uma função com um determinado nome e quantidade de parâmetros;
- Criação de uma função com ou sem retorno;
- Criação de uma função recursiva;
- Exigir a utilização de uma função da API da linguagem de programação;
- Proibir a utilização de uma função da API da linguagem de programação;
- Exigir a utilização de *array*/matriz na solução.

Portanto, com base nos três conjuntos de dados, é possível confirmar a importância de uma abordagem que permita analisar estaticamente se uma solução satisfaz determinadas restrições estruturais.

3. Trabalhos relacionados

Esta seção apresenta três trabalhos relacionados que permitem que o professor especifique e verifique se a solução do aluno possui determinadas construções de programação. O principal objetivo é analisar o processo que deve ser seguido pelo professor para especificar e checar as restrições nas implementações dos alunos.

3.1. Projekt Tomo

Projekt Tomo [Jerse and Lokar 2018, Jerse and Lokar 2016] é um serviço web para correção automática de problemas de programação. O sistema é utilizado por estudantes dos cursos de Matemática e Física da Universidade de Ljubljana, na Eslovênia. As Listagens 1 e 2 apresentam duas formas diferentes disponibilizadas ao professor para realizar verificações estáticas nas soluções dos alunos.

Listagem 1. Análise estática para solução Octave

```
sol = Check.current_part(''solution'')
if strfind(sol, ''det'')
    Check_error(''Sua solucao nao deveria ter a string det.'');
```

A Listagem 1 está verificando se o aluno utilizou a *string det* na solução e enviando uma mensagem de erro caso encontre.

Listagem 2. Análise estática para solução Python

```
tree = ast.parse(Check.current_part(''solution''))
for node in ast.walk(tree):
  if isinstance(node, ast.Call):
    name = node.func.id
    if name == ''det'':
    Check.error(''A funcao det nao deveria ser usada'')
```

A Listagem 2 faz uma análise mais rebuscada do que simplesmente uma busca por *string*. Nesse caso, o professor está interessado em descobrir se o aluno está chamando alguma função com o nome de "det". Para isso é utilizada a biblioteca **AST** da própria

[2]https://www.dropbox.com/s/qqjxfb2lb9tu6oa/dados_questionario.pdf?dl=0

linguagem de programação Python. Esta abordagem certamente demanda muito trabalho de codificação e exige conhecimento profundo da AST da linguagem de programação. Além disso, a checagem deve ser reescrita para cada linguagem de programação (não é unificada), o que por si só já é um desafio.

3.2. Scheme-robo

Scheme-robo [Saikkonen et al. 2001] é um sistema de avaliação automática de exercícios de programação para a linguagem Scheme utilizado completamente sem interferência humana. O sistema possui um conjunto de funcionalidades que engloba as análises dinâmica e estática de avaliação de código.

Para adicionar um novo problema no Scheme-robo, o professor precisa escrever um arquivo de configuração com expressões Lisp que descreve como a solução deve ser analisada. Esse arquivo geralmente possui entre 20 e 100 linhas e contém os casos de teste, modelos de soluções, palavras-chaves proibidas ou necessárias e outras informações. Uma das formas de analisar uma solução por meio do Scheme-robo é verificar se alguns comandos estão presentes na solução do aluno. Depois de realizada essa busca, o professor pode decidir por validar ou invalidar a solução de acordo com seu propósito.

Listagem 3. Sintaxe para especificação de palavra-chave

```
if(keyword <symbol> ...)
(<points> [<comment> ...])
(<points> [<comment> ...])
```

Listagem 4. Ex. de utilização

```
(if (keyword set!))
(fail ``Nao use set! neste
       exercicio!'')
((0))
```

A Listagem 3 possui a sintaxe que deve ser utilizada para verificar se determinado comando está presente na solução do aluno. Por sua vez, a Listagem 4 possui um exemplo de utilização dessa sintaxe. De acordo com Listagem 4, deve ser procurado o símbolo *set!* na solução submetida e enviado um *feedback* para o aluno se o símbolo for encontrado invalidando a resposta. É importante ressaltar que a palavra-chave *set!* corresponde ao símbolo *set!* no código-fonte do aluno e não a um trecho de *string* que contém a palavra *set!* ou um símbolo longo como *foo-set!*.

Esta abordagem fornece uma granularidade fina ao professor. Por meio dela o professor pode checar diversos aspectos do código do aluno. Entretanto, para isso ele deve utilizar muitas linhas de código-fonte, transformando-se em uma solução pouco prática. Além disso, é notável que esta abordagem possui pouca utilidade prática porque a linguagem Scheme é pouco utilizada em cursos de introdução à programação.

3.3. Portugol Studio

O Portugol Studio é um ambiente para aprender a programar, voltado para os iniciantes em programação que falam o idioma português. Possui uma sintaxe fácil, diversos exemplos e materiais de apoio à aprendizagem. Também possibilita a criação de jogos e outras aplicações [Noschang et al. 2014].

Assim como o Scheme-robo e o Projekt Tomo, o Portugol Studio também possui as análises dinâmica e estática. Entretanto, há uma diferença considerável na maneira como a análise estática foi implementada nesta ferramenta. O Portugol Studio utiliza o conceito de *Tree Walkers* [Pelz 2014, Hodecker 2014], que são algoritmos predefinidos para encontrar determinado padrão na AST da solução do aluno. Desta forma, o professor

não precisa desenvolver um programa para verificar se a solução proposta pelo aluno atende aos seus requisitos, mas apenas selecionar, por meio de uma interface gráfica, o *Tree Walker* que deseja e parametrizá-lo de acordo com suas necessidades.

Os *Tree Walkers* desta ferramenta fazem as seguintes verificações: busca por construções que devem estar presentes no código ou que são proibidas, identifica possíveis laços infinitos, verifica se a solução possui uma determinada quantidade de desvios condicionais, verifica a existência de blocos vazios, verifica se o aluno incrementou o índice do vetor/matriz, verifica se há variáveis não utilizadas, verifica se a solução fez uma entrada de dados após ter manipulado variáveis, verifica se a ordem das variáveis passadas por parâmetro no comando *leia* está em uma ordem diferente durante a saída de dados.

4. Code Umpire

A proposta deste trabalho fornece uma ferramenta para especificação e checagem de restrições (requisitos) relacionadas às construções de várias linguagens de programação. Esta ferramenta deve ser fácil de ser utilizada pelo professor (usuário), não necessitando de que ele possua conhecimento específico sobre a formação da AST de uma solução nem precise programar para realizar as especificações. Além disso, esta abordagem leva em consideração os aspectos comuns das principais linguagens de programação imperativas e orientadas a objetos. Atualmente, o Code Umpire trabalha com soluções Java e Python, todavia uma expansão para novas linguagens pode ser realizada sem alterações em seu *core*, bastando seguir o padrão de projeto criacional *Abstract Factory* utilizado desde a sua concepção.

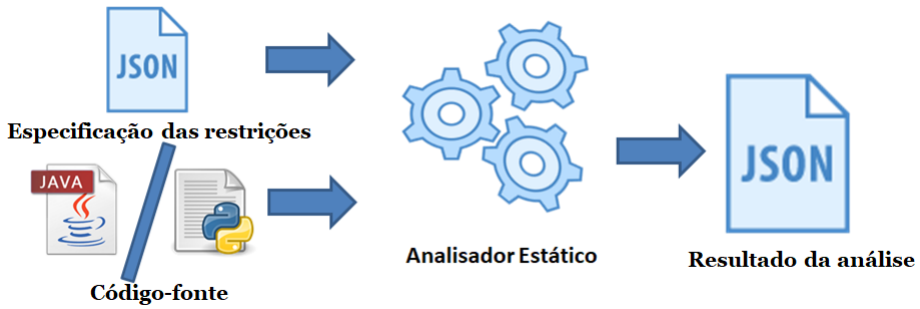

Figura 2. Visão geral da solução

A Figura 2 apresenta o funcionamento da nossa abordagem. No núcleo temos o Analisador Estático que foi implementado em Java com auxílio do *framework* ANTLR 4 [Parr 2013]. O Analisador Estático recebe como entradas o arquivo de especificação de restrições no formato JSON e o código-fonte com a solução do aluno. A saída do Analisador Estático é um arquivo JSON informando se a solução está de acordo com as restrições e se não estiver também informa os problemas encontrados. O manual do Code Umpire com todos os detalhes de formação dos arquivos JSON está disponível no Dropbox[3].

4.1. Exemplo de utilização

Esta subseção exemplifica o funcionamento interno da abordagem proposta. Neste caso, é solicitado ao aluno que desenvolva uma função fibonacci recursiva e com retorno em Python. O aluno deve utilizar exatamente 2 desvios condicionais do tipo *if*.

[3]https://www.dropbox.com/s/0cky08j67ndwyu8/manual_code_umpire.pdf?dl=0

Listagem 5. Solução proposta pelo aluno

```
def fibonacci(num):
    if(num < 2):
        return num
    else:
        return fibonacci(num-1) + fibonacci(num-2)
print(fibonacci(input()))
```

Listagem 6. Especificação das restrições

```
{"restricaoFuncoes":[
  {"nome":"fibonacci",
  "retorno":true,
  "recursiva":true,
  "qtdParametros":1}],

  "restricaoIf":{
  "minimo":2,
  "maximo":2}}
```

Listagem 7. Resultado do analisador estático

```
{"retornoFuncoes":[
  {"existe":true,
  "nome":true,
  "parametros":true,
  "recursao":true,
  "retorno":true,
  "foiChamada":true}],

  "retornoIf":{
  "atendeu":true,
  "quantidade":2}}
```

A Listagem 5 apresenta a solução proposta pelo aluno, neste caso o aluno atendeu a todos os requisitos especificados. A Listagem 6 apresenta o JSON do que foi especificado, este arquivo JSON é gerado automaticamente pela plataforma The Huxley, para inserir as restrições, basta preencher os campos ilustrados na Figura 3. A Listagem 7 possui a saída do analisador estático, este arquivo é interpretado pelo The Huxley que enviará um *feedback* positivo para o aluno.

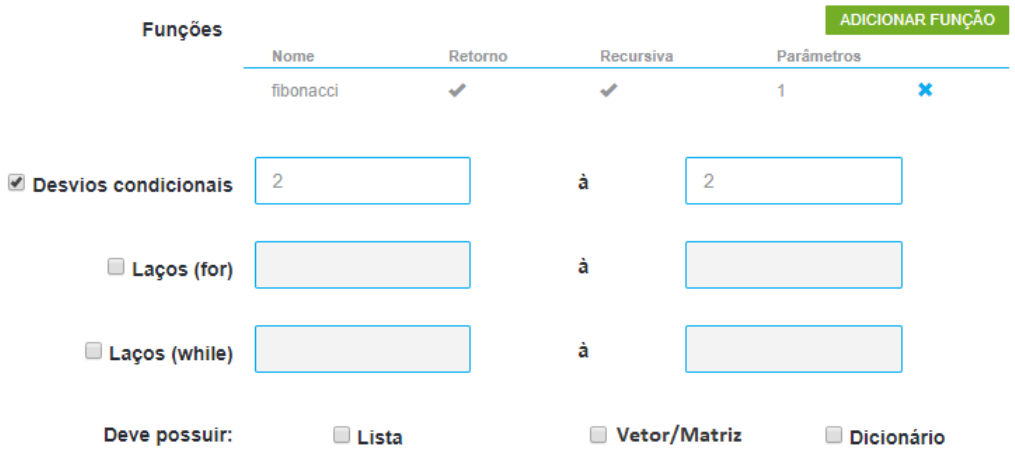

Figura 3. Interface para especificação de restrições

4.2. Grau de flexibilidade

Esta subseção apresenta o grau de flexibilidade da abordagem proposta. Este grau é apresentado na Tabela 2 e está alinhado com o que foi apresentado na Seção 2.

Tabela 2. Tree Walkers implementados

Tree Walkers	**Descrição**
Desvios condicionais	Conta quantos desvios condicionais *if* a solução possui.
While e *For*	Conta quantos *while* e *for* a solução possui.
Função	Verifica se uma função com um determinado nome e quantidade de parâmetros foi criada e invocada. Além disso, permite especificar se a função possui retorno ou deve ser recursiva.
Blocos Vazios	Verifica se há blocos vazios.
Array/Matriz	Verifica se a solução possui declaração de *Array*/Matriz.
Lista e Dicionário	Verifica se a solução possui declaração de lista ou dicionário.
Chamada de função	Verifica se uma função da API foi invocada.

5. Comparação

A Tabela 3 apresenta os critérios de comparação entre os trabalhos relacionados apresentados na Seção 3 e a abordagem desenvolvida neste trabalho.

Tabela 3. Comparação entre as abordagens

Plataforma	**Linguagem**	**Facilidade de uso**	**Grau de flexibilidade**	**Abordagem unificada**
Scheme-robo	Scheme	baixa	elevado	não
Projekt Tomo	Octave/ Python	baixa	elevado	não
Portugol Studio	Portugol	alta	baixo	não
Code Umpire	Java/ Python	alta	adequado	sim

5.1. Facilidade de uso

A facilidade de uso diz respeito ao esforço despendido pelo professor para especificar as restrições estruturais. As abordagens Projekt Tomo e Scheme-robo estão classificadas com uma baixa facilidade de uso porque em ambas o professor precisa dispor de recursos de programação. Esta exigência não é trivial, pois a tarefa de avaliação estática requer um compromisso do professor de imaginar e cercar, por meio de uma solução programática, os diversos cenários que a solução do aluno pode ter.

A plataforma Portugol Studio, assim como a abordagem desenvolvida neste trabalho, não possui o problema da dificuldade de uso apresentado nas outras plataformas. Nessas abordagens, foi desenvolvida a ideia de criação de *Tree Walkers* com comportamentos padronizados que apenas necessitam de parâmetros fornecidos pelo professor para que a avaliação estática seja realizada.

5.2. Grau de flexibilidade

O grau de flexibilidade pode ser definido como o conjunto de restrições que é disponibilizado para que o professor utilize. As abordagens Scheme-robo e Projekt Tomo possuem um elevado grau de flexibilidade porque oferecem ao professor a opção de programar como a solução deve ser avaliada.

Portugol Studio possui um grau de flexibilidade baixo porque não oferece um grande conjunto de restrições de acordo com o levantamento apresentado na Seção 2. Esta abordagem, por exemplo, não permite ao professor especificar que um problema deve ser resolvido com uma função recursiva e um laço de repetição ao mesmo tempo.

A abordagem desenvolvida neste trabalho possui um grau de flexibilidade **adequado** porque está alinhada com a maior parte das restrições levantadas na Seção 2. Para

melhor compreender este aspecto, é necessário entender que há um conjunto predefinido de restrições que geralmente é utilizado pelos professores, esse conjunto foi apresentado na Seção 2. Portanto, nem sempre é viável disponibilizar uma abordagem complexa que seja difícil de ser utilizada pelo professor simplesmente para que ele possa entrar em todos os detalhes da solução proposta pelo aluno. Na verdade, este trabalho propõe uma abordagem simples, fácil de ser utilizada pelo professor e que abrange um conjunto de restrições que é utilizado com frequência.

5.3. Abordagem unificada

Este critério diz respeito à forma como o modelo de avaliação estática foi projetado em cada abordagem. Uma abordagem é considerada unificada se o seu modelo foi concebido para trabalhar eficientemente com várias linguagens de programação. No contexto dos juízes *on-line*, geralmente mais de uma linguagem de programação são disponibilizadas para o aluno e uma abordagem unificada apresenta-se como mais adequada.

A abordagem Scheme-robo não é considerada unificada porque seu modelo de avaliação estática é personalizado para a linguagem funcional Scheme. A abordagem Projekt Tomo também não é considerada unificada porque possui mais de um modelo de avaliação estática que utiliza os recursos da linguagem de programação. Isso fere o conceito de abordagem unificada porque uma linguagem pode oferecer mais recursos do que outra. A abordagem Portugol Studio não pode ser considerada completamente unificada porque em alguns casos demanda do professor conhecimentos específicos da linguagem Portugol. Um exemplo disso é a necessidade de conhecer os nós gerados pela AST de uma solução Portugol para exigir ou proibir determinado comando.

A abordagem proposta neste artigo foi projetada para suportar várias linguagens de programação e não apenas Python e Java. Desta forma, quando a abordagem evoluir e passar a aceitar novas linguagens, o modelo de especificação não sofrerá alteração, pois ele é baseado em um arquivo JSON que não inclui detalhes de determinada linguagem e aceita restrições aplicáveis à maioria das linguagens de programação imperativas e orientadas a objetos. Desta forma, se o professor sabe especificar as restrições com a versão atual que está disponível para Python e Java, ele também saberá para versões futuras que incluam outras linguagens de programação.

6. Conclusão

Este trabalho apresentou uma abordagem para especificar e checar restrições de código-fonte para soluções de problemas de programação. Para o desenvolvimento e validação desta abordagem, foi necessário identificar quais restrições de código-fonte eram mais significativas. Esta verificação foi feita através de uma análise da base de dados do The Huxley e da aplicação de um questionário.

As restrições relacionadas à utilização/proibição de *array*/matriz e funções são as mais utilizadas pelos professores. A restrição de criação e utilização de funções recursivas foi a menos respeitada pelos alunos. Além de servir como base para o desenvolvimento da abordagem, esta análise é considerada fundamental e constitui uma importante contribuição, uma vez que em nenhum trabalho abordado nesta pesquisa foi encontrado um levantamento como este.

Após o desenvolvimento da abordagem, foi realizada uma comparação com outras e percebido que apesar da nossa proposta não ter o maior grau de flexibilidade, ela

atende às principais necessidades dos professores apresentadas na Seção 2. Além disso, prescinde de uma solução programática exigida pelo Scheme-robo e Projekt Tomo e faz uso de um método unificado que serve para diversas linguagens de programação.

O Code Umpire está integrado ao The Huxley, mas sua arquitetura é totalmente independente, o que permite sua integração em outros juízes *on-line*, ferramentas de avaliação automática ou até ser disponibilizado via *Web Services*.

Referências

Ala-Mutka, K. M. (2005). A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102.

Bez, J. L., Tonin, N. A., and Rodegheri, P. R. (2014). URI Online Judge Academic: A Tool for Algorithms and Programming Classes. *The 9th International Conference on Computer Science & Education (ICCSE 2014)*, (Iccse):149–152.

de Barros Paes, R., Malaquias, R., Guimarães, M., and Almeida, H. (2013). Ferramenta para a avaliação de aprendizado de alunos em programação de computadores. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, volume 2.

Hodecker, A. (2014). *Aprimoramento e avaliação do corretor de questões do Portugol Studio*. Trabalho técnico-científico de conclusão de curso, Universidade do Vale do Itajaí.

Jerse, G. and Lokar, M. (2016). Using system for automatic assessment projekt tomo in learning and teaching numerical methods. *CADGME- Conference on Digital Tools in Mathematics Education*.

Jerse, G. and Lokar, M. (2018). Providing better feedback for students solving programming tasks using project tomo. *ISEE (Innovative Software Engineering Education)*.

Noschang, L. F., Fillipi Pelz, E. A., and Raabe, A. (2014). Portugol studio: Uma ide para iniciantes em programaçao. *Anais do CSBC/WEI*, pages 535–545.

Parr, T. (2013). *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.

Pelz, F. D. (2014). *Um gerador de dicas para guiar novatos na aprendizagem de programação*. Dissertação (mestrado em computação aplicada), Universidade do Vale do Itajaí.

Rahman, K. A. and Nordin, M. J. (2007). A review on the static analysis approach in the automated programming assessment systems. *In National conference on programming*.

Saikkonen, R., Malmi, L., and Korhonen, A. (2001). Fully automatic assessment of programming exercises. In *ACM Sigcse Bulletin*, volume 33, pages 133–136. ACM.

Sun, H. and Bofang Li, M. J. (2014). YOJ: An online judge system designed for programming courses. In *2014 9th International Conference on Computer Science & Education*, number Iccse, pages 812–816. IEEE.

Wu, J. and Shuangping Chen, R. Y. (2012). Development and application of online judge system. In *2012 International Symposium on Information Technologies in Medicine and Education*, volume 1, pages 83–86. IEEE.