

UNIVERSIDADE FEDERAL FLUMINENSE

VITOR GAMA LEMOS

# ProspectiveProv: Proveniência Prospectiva de Experimentos Representados por Scripts

NITERÓI

2021

UNIVERSIDADE FEDERAL FLUMINENSE

VITOR GAMA LEMOS

# ProspectiveProv: Proveniência Prospectiva de Experimentos Representados por Scripts

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Orientador:  
Vanessa Braganholo Murta

NITERÓI

2021

Ficha catalográfica automática - SDC/BEE  
Gerada com informações fornecidas pelo autor

L555p Lemos, Vitor Gama  
ProspectiveProv: Proveniência Prospectiva de Experimentos  
Representados por Scripts / Vítor Gama Lemos ; Vanessa  
Braganholo, orientadora. Niterói, 2021.  
87 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense,  
Niterói, 2021.

DOI: <http://dx.doi.org/10.22409/PGC.2021.m.13312292786>

1. Script. 2. Proveniência. 3. Visualização de  
proveniência. 4. Diagramas. 5. Produção intelectual. I.  
Braganholo, Vanessa, orientadora. II. Universidade Federal  
Fluminense. Instituto de Computação. III. Título.

CDD -

# VITOR GAMA LEMOS

ProspectiveProv: Proveniência Prospectiva de Experimentos Representados por Scripts

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Aprovada em Julho de 2021.

## BANCA EXAMINADORA

Vanessa Braganholo Murta

Profa. Vanessa Braganholo Murta - Orientadora, UFF

João Felipe N. Pimentel

D.Sc. João Felipe Nicolaci Pimentel, UFF

Troy Costa Kohwalter

Prof. Troy Costa Kohwalter, UFF

Gleiph G. L. Menezes

Prof. Gleiph Ghiotto Lima de Menezes, UFJF

Niterói

2021

# Resumo

Recentemente, muitas linguagens de programação se popularizaram no ambiente científico, especialmente as linguagens de *script*, devido ao seu alto nível de abstração. Nesse contexto, muitos cientistas começaram a modelar seus experimentos científicos usando linguagens de script como forma de garantir um maior controle e gerenciamento, além de agilizar o processo pela busca de resultados. Se por um lado os *scripts* permitem otimizar o tempo e automatizar os experimentos científicos computacionais, por outro lado, a necessidade de modificações no *script* e de diversas execuções para comprovar ou refutar a hipótese experimental geram um grande volume de dados. A proveniência, nesse contexto, tem papel fundamental para ajudar o cientista a manter o rastro de todas essas mudanças e execuções. No entanto, as ferramentas atuais de gerência de proveniência de experimentos modelados como *scripts* falham em dois aspectos: fornecer uma visualização intuitiva da proveniência prospectiva, e unir proveniência prospectiva e retrospectiva no mesmo diagrama. Para cobrir essas lacunas, esta dissertação apresenta o ProspectiveProv, uma abordagem que visa prover suporte para ajudar os cientistas a entender a estrutura de experimentos de *script*. Para tal, o ProspectiveProv utiliza dados de proveniência prospectiva e retrospectiva para gerar diagramas que representam a estrutura do código do experimento, como forma de apoiar o processo de compreensão e análise de resultados de experimentos científicos modelados como *scripts Python*. Para avaliar a abordagem proposta, comparamos os diagramas gerados pelo ProspectiveProv com os gerados por abordagens semelhantes, e também com o uso isolado de *scripts*. Os resultados experimentais mostram que o ProspectiveProv é tão eficaz e eficiente quanto as abordagens existentes para *scripts* complexos, e que usuários de outras áreas (que não computação) se sentem mais confortáveis ao usar os diagramas gerados pelo ProspectiveProv.

**Palavras-chave:** Script, Proveniência, Visualização de proveniência, diagramas.

# Abstract

Recently, many programming languages have become popular in the scientific environment, especially scripting languages, due to their high abstraction level. In this context, many scientists began to model their scientific experiments in scripting languages to ensure more control and efficient data management when seeking for results. If, on one side, scripts allow scientists to optimize and automate their computational experiments, on the other side, the need to modify the script and execute it several times to confirm or refute the experimental hypothesis generates large quantities of data. Provenance plays a key role in helping scientists keep track of all these changes and execution data in this scenario. However, current provenance management tools for scripts fail in two main aspects: (i) they fail in providing an intuitive visualization of prospective provenance, and (ii) they fail in merging prospective and retrospective provenance in a single diagram. To fill in these gaps, this dissertation presents ProspectiveProv, an approach that aims at helping scientists to understand the structure of script experiments. To do so, ProspectiveProv uses prospective and retrospective provenance to generate diagrams that represent the structure of the experiment code as a means to support the process of understanding and analyzing the results of scientific experiments modeled as Python scripts. To evaluate our proposed approach, we compare the diagrams generated by ProspectiveProv with those generated by similar approaches, and also with the isolated use of scripts (no diagrams). The experimental results show that ProspectiveProv is as efficient and effective as the compared approaches when considering complex scripts. Also, users from other knowledge areas (not computer science) felt more comfortable when using diagrams generated by ProspectiveProv.

**Keywords:** Script, Provenance, Provenance visualizations, Diagrams.

# Listas de Figuras

2.1	Ciclo de vida de um experimento científico [40]. . . . .	7
3.1	Passo-a-passo da coleta de dados do Sumatra [14]. . . . .	14
3.2	Grafo de proveniência gerado pelo RDATATracker [37]. . . . .	16
3.3	Grafo de proveniência do ProvenanceCurious [31]. . . . .	17
3.4	Arquitetura do ProvenanceCurious [32]. . . . .	18
3.5	Arquitetura do noWorkflow [53]. . . . .	19
3.6	Grafo de ativação gerado pelo noWorkflow [44]. . . . .	20
3.7	Dataflow gerado após execução do <i>script</i> de exemplo [53]. . . . .	20
3.8	Script de exemplo para gerar um gráfico de temperatura e precipitação durante um período de tempo específico [53]. . . . .	21
3.9	Exemplo de anotações na ferramenta YesWorkflow [42]. . . . .	22
3.10	Grafo de proveniência gerado pelo YesWorkflow [42]. . . . .	23
3.11	Interface para edição e criação de diagramas no Visustin [46]. . . . .	24
3.12	Arquitetura do No+YesWorkflow [15]. . . . .	26
4.1	Arquitetura do ProspectiveProv. . . . .	30
4.2	Modelo de dados do noWorkflow [49] . . . . .	31
4.3	Módulo de coleta de dados de proveniência. . . . .	32
4.4	Lista de consultas utilizadas pelo ProspectiveProv para consultar os dados de proveniência. . . . .	33
4.5	Rotulação dos nós do grafo de proveniência. . . . .	34
4.6	Gerando e ligando novos nós do grafo de proveniência. . . . .	35
4.7	Simbologia utilizada para representar os nós no ProspectiveProv. . . . .	35

4.8	Diagrama de proveniência prospectiva produzido pelo ProspectiveProv. . . . .	36
4.9	Diagrama com a união da proveniência prospectiva e retrospectiva produzido pelo ProspectiveProv. . . . .	37
4.10	Ligando dados de proveniência retrospectiva e prospectiva. . . . .	38
4.11	Diagrama de proveniência prospectiva. . . . .	39
4.12	Diagrama de proveniência prospectiva (com endentação). . . . .	40
4.13	Diagrama de proveniência prospectiva e retrospectiva (somente conteúdo das variáveis). . . . .	40
4.14	Diagrama de proveniência prospectiva e retrospectiva (somente ativações). .	41
4.15	Diagrama de proveniência prospectiva (parte do código). . . . .	41
4.16	Diagrama de proveniência prospectiva (exibindo apenas uma função). . . .	42
5.1	Fluxograma relacionado aos processos envolvidos no experimento. Tarefas $A_1$ , $A_2$ e $A_3$ são tarefas da Fase 1 do experimento, enquanto as tarefas $B_1$ e $B_2$ pertencem à Fase 2. . . . .	48
5.2	Fluxograma relacionado aos processos envolvidos numa fase. . . . .	49
5.3	Participantes divididos por experiência acadêmica. . . . .	51
5.4	Participantes divididos por área do conhecimento. . . . .	52
5.5	Participantes divididos por experiência em Python. . . . .	52
5.6	Participantes divididos por experiência em software. . . . .	53
5.7	Taxa de respostas corretas. . . . .	54
5.8	Análise da variável duração (Fase 1). . . . .	56
5.9	Análise da variável duração (Fase 2). . . . .	57
5.10	Grau de dificuldade na execução da tarefa $A_1$ . . . . .	59
5.11	Grau de dificuldade na execução da tarefa $A_2$ . . . . .	59
5.12	Grau de dificuldade na execução da tarefa $A_3$ . . . . .	59
5.13	Grau de dificuldade na execução da tarefa B. . . . .	60
5.14	Preferência por abordagens (comparação entre <i>scripts</i> e diagramas). . . .	61

5.15 Teste de corretude utilizando laços condicionais. . . . .	62
5.16 Teste de corretude utilizando <i>loops</i> . . . . .	63
5.17 Teste de corretude utilizando funções. . . . .	64
D.1 Script de exemplo utilizado no experimento . . . . .	82
D.2 Diagrama de exemplo utilizado no experimento . . . . .	83
E.1 Questionário de Avaliação I . . . . .	84
F.1 Questionário de Avaliação II . . . . .	85

# Listas de Tabelas

3.1	Sistemas para Captura de Proveniência.	27
5.1	Conhecimento em Python necessário para conclusão das tarefas da Fase 1.	49
5.2	Esquema de quadrados latinos (Fase 1).	49
5.3	Esquema de quadrados latinos (Fase 2).	50
5.4	Conhecimento em Python necessário para conclusão das tarefas da Fase 2.	51
5.5	<i>P-value</i> para o teste de Fisher para o número total de respostas corretas. A hipótese nula é que não há diferença entre as abordagens comparadas. Valores azuis indicam <i>p-value</i> $\leq \alpha$ , com $\alpha = 0.05$ .	55
5.6	<i>P-value</i> para o teste de Mann-Whitney para a variável duração. A hipótese nula é que não há diferença entre as abordagens comparadas. Os valores azuis indicam <i>p-value</i> $\leq \alpha$ , com $\alpha = 0.05$ .	57
5.7	Nível de dificuldade com base nas abordagens avaliadas (Fase 1).	60
5.8	Nível de dificuldade com base nas abordagens avaliadas (Fase 2).	61
A.1	Lista de diagramas recebidos pelos participantes conforme o esquema de quadrados Latinos.	74
A.2	Detalhe do que se pede em cada uma das questões	74
B.1	Lista de diagramas recebidos pelos participantes conforme o esquema de quadrados Latinos.	75

# Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Motivação . . . . .	1
1.2	Objetivos . . . . .	3
1.3	Metodologia . . . . .	4
1.4	Contribuições . . . . .	4
1.5	Organização . . . . .	5
<b>2</b>	<b>Proveniência em <i>scripts</i></b>	<b>6</b>
2.1	Introdução . . . . .	6
2.2	Experimentos Científicos . . . . .	6
2.3	Proveniência . . . . .	8
2.4	Considerações Finais . . . . .	11
<b>3</b>	<b>Trabalhos Relacionados</b>	<b>13</b>
3.1	Introdução . . . . .	13
3.2	Coleta e Visualização de Proveniência . . . . .	13
3.2.1	Sumatra . . . . .	13
3.2.2	RDataTracker . . . . .	14
3.2.3	ProvenanceCurious . . . . .	16
3.2.4	noWorkflow . . . . .	18
3.2.5	YesWorkflow . . . . .	22
3.2.6	Visustin . . . . .	24

3.3	Ligaçāo de Proveniēncia Prospectiva e Retrospectiva . . . . .	25
3.4	Considerações Finais . . . . .	26
<b>4</b>	<b>ProspectiveProv</b>	<b>28</b>
4.1	Introdução . . . . .	28
4.2	Arquitetura . . . . .	29
4.3	Acesso a Dados de Proveniēncia . . . . .	30
4.4	Mapeamento dos Nós do Grafo . . . . .	31
4.5	Visualizaçāo da Proveniēncia . . . . .	33
4.6	Conectando Proveniēncia Prospectiva e Retrospectiva . . . . .	35
4.7	Uso do ProspectiveProv . . . . .	38
4.8	Considerações Finais . . . . .	42
<b>5</b>	<b>Estudo Experimental</b>	<b>43</b>
5.1	Introdução . . . . .	43
5.2	Design do Experimento . . . . .	44
5.2.1	Fase 1: Comparaçāo das abordagens do estudo . . . . .	47
5.2.2	Fase 2: Comparaçāo do uso de scripts e diagramas . . . . .	50
5.2.3	Perfil dos participantes . . . . .	50
5.3	Resultados e Discussões . . . . .	52
5.3.1	Eficácia . . . . .	53
5.3.2	Eficiêncāa . . . . .	56
5.3.3	Questionário de avaliação dos participantes . . . . .	58
5.4	Avaliaçāo de corretude . . . . .	60
5.5	Ameaças a validade . . . . .	63
5.6	Considerações Finais . . . . .	65
<b>6</b>	<b>Conclusāo</b>	<b>66</b>

6.1 Resultados . . . . .	66
6.2 Limitações . . . . .	67
6.3 Trabalhos Futuros . . . . .	68
 <b>Referências</b>	 69
 <b>Apêndice A – Mapeamento do Experimento (Fase 1)</b>	 74
 <b>Apêndice B – Mapeamento do Experimento (Fase 2)</b>	 75
 <b>Apêndice C – Lista de Imagens (Fase 1)</b>	 76
 <b>Apêndice D – Lista de Imagens (Fase 2)</b>	 82
 <b>Apêndice E – Questionário de Avaliação 1</b>	 84
 <b>Apêndice F – Questionário de Avaliação II</b>	 85
 <b>Apêndice G – Termo de Consentimento Livre e Esclarecido (TCLE)</b>	 86

# Capítulo 1

## Introdução

### 1.1 Motivação

Com a origem da computação, muitas das pesquisas científicas que antes eram apenas teorias puderam ser comprovadas. Isso permitiu avanços nunca antes imaginados. Sem dúvida, é evidente que a tecnologia desempenha um papel fundamental ao possibilitar a medição de fenômenos que antes eram inacessíveis à pesquisa científica [64]. Como exemplo disso, podemos citar as mais diversas pesquisas científicas desenvolvidas apenas graças a métodos computacionais [2]. Além desses trabalhos, recentemente, com ajuda de ferramentas e métodos sofisticados providos pela computação científica, foi possível capturar a primeira imagem real de um buraco negro [2], o que comprovou os trabalhos de Albert Einstein sobre a Teoria da Relatividade Geral [19].

A evolução tecnológica também permitiu aos cientistas automatizar a execução de seus próprios experimentos científicos. Por esse motivo, muitas linguagens de programação se popularizaram no meio científico, principalmente as linguagens de *script*, devido ao seu altíssimo nível de abstração, como é o caso do Python<sup>1</sup>, R<sup>2</sup> e MATLAB<sup>3</sup>. Nesse aspecto, muitos cientistas começaram a modelar seus experimentos científicos em linguagens de *script* devido à facilidade de configuração e execução, sendo Python a linguagem mais utilizada [49]. Na literatura, muitos trabalhos científicos utilizam linguagens de *script* para modelagem de experimentos científicos em diversas áreas do conhecimento, tais como na bioinformática [55, 25], *machine learning* [8, 47, 1], medicina [41], oceanografia [39, 54, 56], entre outros.

---

<sup>1</sup><https://www.python.org>

<sup>2</sup><https://www.r-project.org>

<sup>3</sup><https://www.mathworks.com/products/matlab.html>

Se por um lado os *scripts* permitem otimizar o tempo e automatizar os experimentos científicos computacionais, por outro lado, falhas e inconsistências na construção do código podem obrigar o cientista a realizar constantes validações e re-execuções. Diante desse ponto, a questão é, como podemos garantir a qualidade, confiabilidade e reproduzibilidade dos resultados se não conseguimos interpretá-los? A resposta dessa e de muitas outras perguntas pode estar na proveniência, uma das soluções que tem ganhado alta relevância no âmbito científico nos últimos anos [9, 28, 29, 24].

A proveniência descreve todos os processos envolvidos na produção de um produto específico [29]. Na computação, a proveniência é uma importante aliada, principalmente para garantir a confiabilidade e reprodução dos resultados. A captura e o gerenciamento dela é muito relevante e possui diversas aplicações [24]. Dessa forma, diversas ferramentas foram projetadas para dar apoio a coleta e visualização de proveniência de experimentos materializados como *script*. Dentre os trabalhos que focam na coleta, gerenciamento e visualização de proveniência podemos citar o noWorkflow [44, 53], YesWorkflow [42], No+YesWorkflow [15], ProvenanceCurious [32, 31], RDataTracker [37], entre outros.

No que se refere a proveniência de *scripts*, ela pode ser classificada em proveniência retrospectiva ou proveniência prospectiva [24, 29, 38]. A proveniência prospectiva é relacionada à dados da estrutura do código, enquanto a proveniência retrospectiva inclui informações relacionadas a execução dos *scripts*. Recentemente, diversas ferramentas surgiram com a capacidade de coletar proveniência de experimentos representados por *scripts*. No entanto ainda há lacunas a serem cobertas, principalmente no que se refere a proveniência prospectiva, que é o foco desse trabalho.

Para identificar essas lacunas, foi realizada uma revisão da literatura com o propósito de identificar as principais características das abordagens que permitem coletar e visualizar a proveniência prospectiva de *scripts*. Este estudo também observou que as principais abordagens para visualização de proveniência em *scripts* não permitem unir proveniência prospectiva e retrospectiva em um único diagrama. A única abordagem que permite isso – No+YesWorkflow [15] – ainda necessita do auxílio dos cientistas para criar anotações diretamente nos *scripts*. Essas anotações são então usadas para gerar os grafos de proveniência. O estudo também identificou que muitas dessas abordagens geram grafos usando uma notação muito própria, o que pode dificultar a análise por pessoas não familiarizadas com a notação. Isso pode acarretar dificuldades em chegar a conclusões corretas em relação aos resultados do experimento. Ferramentas como o noWorkflow [44] coletam a proveniência relacionada à estrutura de um *script* (proveniência prospectiva) e também à

sua execução (proveniência retrospectiva). Embora o noWorkflow colete ambos os tipos de proveniência, somente a proveniência retrospectiva é representada visualmente usando grafos de proveniência. Em resumo, as abordagens existentes para capturar, gerenciar e visualizar a proveniência de *scripts* apresentam limitações, nas quais destacam-se:

1. Uso de notações próprias para visualização de proveniência. Dados de proveniência de *scripts* são fundamentais para o entendimento e análise de resultados, e a visualização dela é um ponto essencial para transmissão desse conhecimento. Assim, faz-se necessário uma discussão e comparação da eficiência e efetividade dos métodos para visualização da proveniência prospectiva que usam notações distintas.
2. Dificuldade em ligar proveniências prospectiva e retrospectiva. Tanto os dados da estrutura de um *script* quanto detalhes de sua execução são importantes para facilitar a avaliação de resultados de experimentos. Na literatura, não foram encontrados sistemas que ligam proveniência prospectiva e retrospectiva de *scripts* de forma transparente em uma forma visual única. Embora o No+YesWorkflow [15] consiga combinar proveniência prospectiva e retrospectiva, ele ainda dependente totalmente de anotações feitas no *script* para gerar diagramas de proveniência e, portanto não funciona de forma automática.

## 1.2 Objetivos

No contexto de *scripts*, a visualização de dados de proveniência prospectiva é importante para identificar falhas e melhorar o entendimento sobre o experimento. Entender os resultados e ao mesmo tempo entender qual parte do código está produzindo tais resultados é fundamental para otimizar a forma como os cientistas analisam seus resultados.

O objetivo deste trabalho é preencher as lacunas quanto a visualização de proveniência elencadas na seção anterior. Para atingir este objetivo, neste trabalho propomos o ProspectiveProv, uma abordagem para apoiar os cientistas na compreensão da estrutura dos experimentos estruturados como *scripts* Python usando diagramas. Além disso, o ProspectiveProv permite combinar a proveniência prospectiva e retrospectiva para que os cientistas entendam a relação entre a estrutura de um *script* e os resultados produzidos em cada parte dele.

Para atingir esses objetivos, o ProspectiveProv aproveita dados de proveniência coletados pelo noWorkflow [44, 53] para criar diagramas que permitem tanto exibir dados

da estrutura do *script* quanto de detalhes de sua execução. O noWorkflow (e consequentemente, o ProspectiveProv) é capaz de capturar dados de proveniência de experimentos materializados na forma de *scripts* Python, o que se justifica por sua ampla adoção pela comunidade científica [49].

## 1.3 Metodologia

Para avaliar a abordagem desse estudo e identificar possíveis limitações, foi realizado um estudo experimental dividido em duas fases. Na Fase 1, comparamos os diagramas gerados pelo ProspectiveProv com os diagramas de abordagens similares existentes na literatura. Na Fase 2 comparamos os diagramas produzidos pelo ProspectiveProv com os *scripts* originais. Para isso, foram elaboradas as seguintes questões de pesquisa.

Com relação a **eficácia**:

- **Questão de Pesquisa 1.** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficazes na análise e compreensão da estrutura de um *script* em comparação com outras abordagens com este mesmo propósito?
- **Questão de Pesquisa 2.** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficazes na análise e compreensão da estrutura de um *script* em comparação com não usar essa abordagem (usar apenas o código do script)?

Com relação a **eficiência**:

- **Questão de Pesquisa 3.** Os diagramas ProspectiveProv permitem que os usuários sejam mais eficientes na análise e no entendimento da estrutura de um *script* em comparação com outras abordagens com este mesmo propósito?
- **Questão de Pesquisa 4.** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficientes na análise e no entendimento da estrutura de um *script* em comparação com não usar essa abordagem (usar apenas o código do script)?

## 1.4 Contribuições

As contribuições desse trabalho são:

- A abordagem ProspectiveProv para apoiar a compreensão de experimentos desenvolvidos como *scripts* Python utilizando dados de proveniência.
- Uma ferramenta (disponível como código aberto no GitHub<sup>4</sup>) que usufrui de dados de proveniência prospectiva e retrospectiva para produzir diagramas úteis e simplificados para facilitar a análise dos cientistas em seus experimentos de *scripts*.
- A identificação de pontos de melhoria para as ferramentas existentes com relação a visualização de proveniência, o que possibilita o aprimoramento interno do ProspectiveProv.

A partir da análise dos resultados da avaliação experimental, a abordagem aqui proposta apresentou os seguintes resultados:

- Mais eficiência e eficácia na compreensão dos diagramas quando comparado ao ProvenanceCurious [32].
- 60% dos participantes de áreas diferentes da computação se sentiram mais confortáveis no uso de diagramas do ProspectiveProv quando comparado ao uso apenas código fonte dos *scripts*.

## 1.5 Organização

Esta dissertação está organizada em outros cinco capítulos, além desta introdução. O Capítulo 2 apresenta conceitos essenciais sobre proveniência em seu âmbito geral e também fundamentos relacionados a proveniência de tarefas computacionais. Além disso, apresenta casos de uso para a proveniência no campo científico e de experimentos computacionais. O Capítulo 3 discute os trabalhos relacionados a este estudo. Também são analisadas ferramentas destinadas a coleta e visualização de proveniência de tarefas computacionais. O Capítulo 4 apresenta o ProspectiveProv. Inicialmente, é apresentada uma visão geral da abordagem. As seções seguintes descrevem arquitetura e os métodos utilizados para gerar e ligar diferentes tipos de proveniência. O Capítulo 5 descreve o estudo experimental realizado para comparar a abordagem proposta neste trabalho com abordagens existentes na literatura. São realizadas análises estatísticas para avaliar a eficácia e eficiência das abordagens analisadas. Também são realizadas discussões de resultados e detalhes sobre as questões de pesquisa. Finalmente, o Capítulo 6 apresenta as conclusões desta dissertação e propõe trabalhos futuros.

---

<sup>4</sup><https://github.com/dew-uff/prospective-prov>

# Capítulo 2

## Proveniência em *scripts*

### 2.1 Introdução

Este capítulo apresenta detalhes referentes a proveniência de *scripts* e seus principais conceitos, de maneira a expor a fundamentação teórica necessária para a compreensão dos assuntos abordados nessa dissertação. A Seção 2.2 conceitua experimentos científicos. A Seção 2.3 descreve os principais conceitos relacionados a proveniência. Por fim, a Seção 2.4 apresenta as considerações finais deste capítulo.

### 2.2 Experimentos Científicos

Os experimentos são fundamentais para a investigação científica. Levando em consideração as etapas do método científico, os experimentos são responsáveis por verificar se a hipótese levantada é realmente verdadeira. Utilizando uma definição mais formal, os experimentos científicos são testes realizados sob condições controladas, o que permite ao investigador o controle e manipulação dos parâmetros de entrada, a fim de capturar a causalidade do fenômeno de estudo [36]. Portanto, podemos dizer que o estudo experimental é o processo em que o investigador (pesquisador) precisa observar, avaliar, perguntar, interpretar e comparar resultados.

Um experimento científico é composto por várias etapas que compõem o seu ciclo de vida. De acordo com Mattoso et al. [40], o ciclo de vida de um experimento é composto por três fases principais (ver Figura 2.1). Primeiro, a fase de composição é a responsável pela estruturação de todo o experimento, estabelecendo a sequência lógica de passos a serem executados e as configurações do experimento científico. A segunda fase,

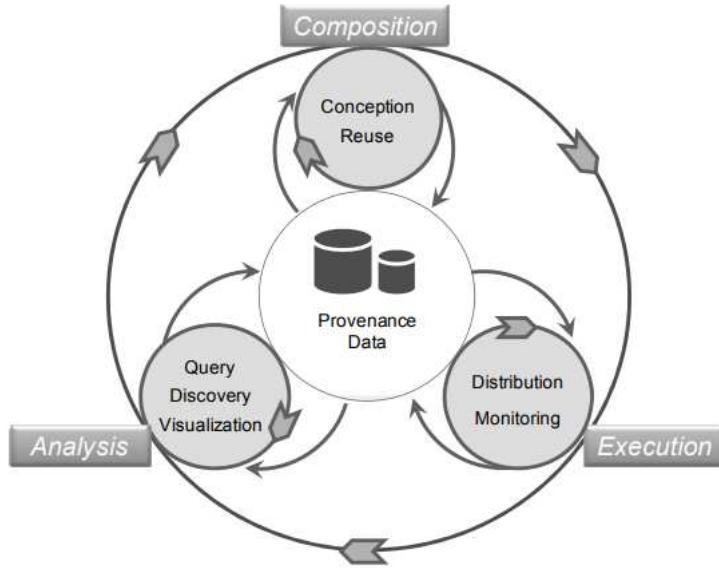


Figura 2.1: Ciclo de vida de um experimento científico [40].

chamada de execução, é a responsável pela execução do experimento. Isso também inclui o armazenamento dos dados de entrada e saída referentes ao experimento. Por fim, a última parte da sequência, a fase de análise, é a responsável pela investigação e análise dos resultados obtidos durante as etapas anteriores, podendo o cientista recomeçar um novo ciclo.

Os experimentos científicos podem ser conduzidos em laboratórios, em ambientes reais ou simulados. Atualmente, uma grande parte das pesquisas científicas são realizadas em ambientes virtuais ou simulados – são os estudos *in silico* ou *in virtuo*. De acordo com Travassos e Brarros [63], um experimento científico pode ser categorizado em quatro grupos: os estudos *in vivo*, *in vitro*, *in virtuo* e *in silico*. Tais experimentos são caracterizados da seguinte maneira:

- Os estudos *in vivo* são totalmente realizados em ambientes reais, envolvendo a participação de pessoas ou de outros objetos de estudo. Em tais experimentos, não existem modelos de simulação computacional envolvidos. Além disso, os estudos *in vivo* custam mais caro do que qualquer outro e existe um alto risco de ameaças à validade dos experimentos. Nesse caso, replicar um experimento *in vivo* se torna complicado e caro.
- Os estudos *in vitro* são conduzidos em ambientes mais controlados, como laboratórios de pesquisa [63]. Nesses estudos, o pesquisador possui um certo nível de controle quanto ao ambiente de testes e também de fatores que podem afetar os

resultados do experimento.

- Os estudos *in virtuo* são realizados em ambientes virtuais, utilizando modelos computacionais para simular cenários reais. Isso exige que esses estudos sejam realizados por meio do computador, mas envolvendo também a participação de pessoas, seja manipulando algum software específico ou simulador. Esse tipo de estudo é caracterizado por pequenos grupos manipulando simuladores em laboratórios ou universidades [63].
- Os estudos *in silico* são totalmente virtuais, sem qualquer interação humana. Logo, tais experimentos não envolvem cenários reais. Tudo é simulado por computador, inclusive a participação de pessoas, que é substituída por agentes inteligentes ou algoritmos autônomos. Por essa razão, estudos *in silico* são os que mais produzem dados para analisar e armazenar, sendo dependentes de recursos computacionais de alto desempenho.

Como meio de facilitar a análise de experimentos realizados como tarefas computacionais, os *workflows* científicos são utilizados como abstrações para modelagem de fluxos de trabalho, representando os passos de execução de um determinado experimento científico [24, 40]. Os *workflows* são responsáveis por descrever as atividades a serem executadas em um experimento. Com relação a proveniência, os Sistemas de Gerência de Workflows podem capturar processos complexos em vários níveis de detalhes e registrar as informações de proveniência necessárias para a automação, reproduzibilidade e compartilhamento de resultados [24].

No entanto, nessa dissertação, estamos interessados nos experimentos científicos modelados como *scripts* Python, linguagem muito conhecida por sua simplicidade. Essa popularidade está em crescimento [12]. De fato, existem inúmeros trabalhos que utilizam essa linguagem de programação para a execução de experimentos em diferentes domínios científicos, como na área de bioinformática [55, 25], computação científica [8, 47, 1], medicina [41], oceanografia [39, 54, 56], entre muitas outras áreas.

## 2.3 Proveniência

A proveniência tem ganhado alta relevância no âmbito científico [9, 51, 24, 28, 29, 61]. Essa seção tenta responder alguma das questões a respeito da proveniência: *O que realmente*

*é a proveniência? Por que usar proveniência? De que modo a proveniência pode nos ajudar?*

A proveniência descreve, de maneira simples, todo o processo de produção de um produto específico [29]. Com isso, podemos dizer que a proveniência nos possibilita entender a origem de um produto ou resultado, levando em consideração a sequência de etapas relacionadas ao seu processo de produção. Embora a proveniência seja empregada na computação, o conceito surgiu inicialmente na arte, uma vez que os especialistas em arte precisavam traçar a origem e o histórico de obras de arte para confirmar sua autenticidade.

A proveniência também pode ser vista como metadados que, ao invés de descrever dados, descrevem processos [29]. Desse modo, a proveniência permite descrever processos complicados de maneira abstrata, identificando informações que levaram a um determinado resultado experimental. Além disso, a proveniência permite que cientistas depositem confiança em seus dados e compreendam como os resultados foram derivados na execução de um experimento científico [11, 57, 58].

A proveniência pode ajudar aos cientistas a descrever seus experimentos. Assim, coletar a proveniência permite compreender os processos envolvidos na pesquisa, dados talvez ignorados pela dificuldade de interpretá-los e analisá-los. Diante disso, são motivos para o uso da proveniência [24, 29]:

- A *reprodutibilidade*, cujo objetivo é garantir a qualidade e confiabilidade de resultados obtidos na execução de experimentos científicos. Conforme Freire [24], Bonnet e Sasha [22] e Freire e Chirigati [23], um experimento científico composto por uma sequência de passos definidos por  $S$ , desenvolvido em um tempo  $T$ , no ambiente  $E$  (hardware e sistema operacional), e usando os dados  $D$  é reproduzível se puder ser executado com uma sequência de etapas  $S'$  (diferente ou igual a  $S$ ), em um tempo  $T' \geq T$ , no ambiente computacional  $E'$  (diferente ou igual a  $E$ ) e usando os dados  $D'$  (diferentes ou iguais a  $D$ ) com resultados consistentes. A reproducibilidade é abordada em muitos trabalhos acadêmicos [7, 11, 18, 35, 48, 58, 59]
- A *identificação de dependências*, essencial na reproducibilidade. A proveniência fornece informações importantes que permitem a identificação de dependências. Em *scripts*, por exemplo, a identificação de dependências pode ajudar na instalação de bibliotecas necessárias para a execução de um mesmo experimento em outros ambientes computacionais.
- A *colaboração*, um importante requisito para o avanço nas descobertas científicas

e na educação] [20, 34]. A proveniência fornece as informações necessárias para a transmissão de dados entre colaboradores de um mesmo projeto, permitindo entender as ações de cada um dentro do grupo de colaboradores ou pesquisadores. Alguns trabalhos abordam a construção de redes colaborativas baseados na proveniência, como por exemplo [20, 43].

- A **visualização de dados**, que permite abstrair dados complexos e representá-los de maneira simples. A proveniência permite extrair e transformar os dados por meio de modelos visuais, facilitando a interpretação e compreensão das etapas envolvidas em um experimento. Existem muitos sistemas que coletam a proveniência e também implementam técnicas para visualização dos dados coletados, como é o caso do Galaxy [27], Vistrails [10], Taverna [30], noWorkflow [44, 53], YesWorkflow [42], entre outros.

**Tipos de proveniência em tarefas computacionais.** O problema de capturar e gerenciar sistematicamente a proveniência de tarefas computacionais é relevante para uma ampla gama de domínios e aplicações [24]. Em tarefas computacionais, a proveniência fornece informações importantes e necessárias para a automação, reproduzibilidade e o compartilhamento de resultados [24]. Quanto a isso, a proveniência pode ser classificada de duas maneiras, em proveniência retrospectiva ou proveniência prospectiva [24, 29, 38].

- A **proveniência retrospectiva** representa a origem dos dados coletados ao longo de um ciclo de execução do experimento. Isso inclui informações referentes a operações executadas e ao ambiente de execução do experimento. A proveniência retrospectiva é coletada com base nas instruções ou processos executados, como por exemplo, as chamadas de função no código de um programa. Assim, com a proveniência retrospectiva, os cientistas podem encontrar, avaliar e compreender os resultados obtidos levando em consideração o que acontece durante a execução de seu experimento. Por ser capturada durante uma execução, ela também fornece informações aos cientistas do comportamento de seu experimento, como tempo de execução de uma linha de código, caso a proveniência retrospectiva seja capturada da execução de um *script*. Muitos sistemas capturam a proveniência retrospectiva, tais como o Hadoop-PROV [4], WebLab PROV [6], VisTrails [10], Ariadne [26], Galaxy [27], Titian [33], noWorkflow [53, 44] e DFL Designer [62].
- A **proveniência prospectiva** representa as informações relacionadas à estrutura de um experimento. Em *scripts*, a proveniência prospectiva representa os dados

relacionados às instruções, como laços condicionais, arquivos de entrada e saída, estruturas de repetição, variáveis globais, parâmetros, e as chamadas de função. Assim, a proveniência prospectiva não requer alguma execução prévia do experimento. Podemos dizer que a proveniência prospectiva ajuda o cientista a entender como um experimento é construído. De um modo mais amplo, grande parte dos sistemas possibilita coletar a proveniência prospectiva e retrospectiva ao mesmo tempo, como é o caso do Kepler [5], Taverna [30] e noWorkflow [44, 53]. Outros sistemas, como o YesWorkflow [42] e o ProvenanceCurious [31, 32] permitem apenas a coleta de proveniência prospectiva.

Como visto anteriormente, a proveniência de tarefas computacionais pode ser dividida em proveniência prospectiva e proveniência retrospectiva. Para a proveniência de *scripts*, utilizamos a nomenclatura definida nos trabalhos de [53, 44], que categoriza a proveniência de *scripts* de três formas, conforme descrito a seguir:

- **Proveniência de definição (*definition*)**. A proveniência de definição é equivalente a proveniência prospectiva. Nesse caso, os sistemas que coletam tal forma de proveniência podem extrair informações sobre a estrutura geral de um *script*, como as variáveis ou funções declaradas, laços de repetição, entre outros. Embora a proveniência de definição não exija uma execução do *script*, também é possível capturá-la durante a execução do código.
- **Proveniência de implantação (*deployment*)**. A proveniência de implantação representa as informações relacionadas ao ambiente de execução, tais como a versão do sistema operacional, valores das variáveis de ambiente, dependências e a arquitetura de *hardware* [44].
- **Proveniência de execução (*execution*)**. A proveniência de execução corresponde a proveniência retrospectiva. Assim, os dados são coletados ao longo da execução do *script*. As informações capturadas incluem as funções ativadas durante a execução, conteúdo de variáveis, valores de retorno e até informações de tempo de execução.

## 2.4 Considerações Finais

Neste capítulo foram apresentados detalhes sobre a proveniência e como ela pode ajudar os cientistas a interpretar e analisar os resultados de seus experimentos de *script*. Além

disso, também foi descrito o ciclo de vida de um experimento científico e como eles são essenciais para comprovar ou refutar uma hipótese de uma pesquisa. O próximo capítulo mostra outros detalhes referentes a proveniência de *scripts* e os trabalhos que já foram desenvolvidos nesse assunto.

# Capítulo 3

## Trabalhos Relacionados

### 3.1 Introdução

Na literatura, muitos trabalhos abordam a coleta de proveniência de *scripts* [15, 24, 29, 42, 44, 53, 32, 31, 57, 58]. De um modo geral, existem muitas abordagens para a coleta de proveniência dedicadas a tarefas computacionais, tanto para a captura de proveniência prospectiva [4, 6, 10, 27, 33, 42] quanto da retrospectiva [5, 30, 44, 53]. De todos os sistemas mencionados, devemos destacar principalmente o noWorkflow [44, 53], YesWorkflow [42], Sumatra [14] e ProvenanceCurious [32, 31], pois tais ferramentas fornecem um ambiente de captura de proveniência direcionado a *scripts* em Python, que é a linguagem alvo da solução proposta nessa dissertação.

Este capítulo está organizado como segue. A Seção 3.2 apresenta as abordagens na literatura para a coleta e visualização de proveniência de tarefas computacionais. A Seção 3.3 apresenta as abordagens relacionadas a capturar e unir a proveniência prospectiva e retrospectiva. Finalmente, a Seção 3.4 apresenta as considerações finais deste capítulo.

### 3.2 Coleta e Visualização de Proveniência

#### 3.2.1 Sumatra

O Sumatra é uma biblioteca em Python que fornece uma interface para coletar dados da execução de um *script*, permitindo também visualizar, pesquisar e anotar registros de execução. O Sumatra tem como objetivo apoiar pesquisas reproduzíveis, que podem ser pensadas como um notebook de laboratório automatizado para projetos computacionais [14]. Na versão atual, o Sumatra fornece uma interface de linha de comando,

capturando entradas e saídas da execução de um *script* Python e fornecendo o registro de cálculos anteriores diretamente do console [14]. Atualmente, o Sumatra possibilita que o usuário acesse seus recursos por meio de uma interface Web, mas não permite gerar grafos de proveniência. A Figura 3.1 mostra as etapas para capturar o contexto de um experimento de *script* usando o Sumatra.

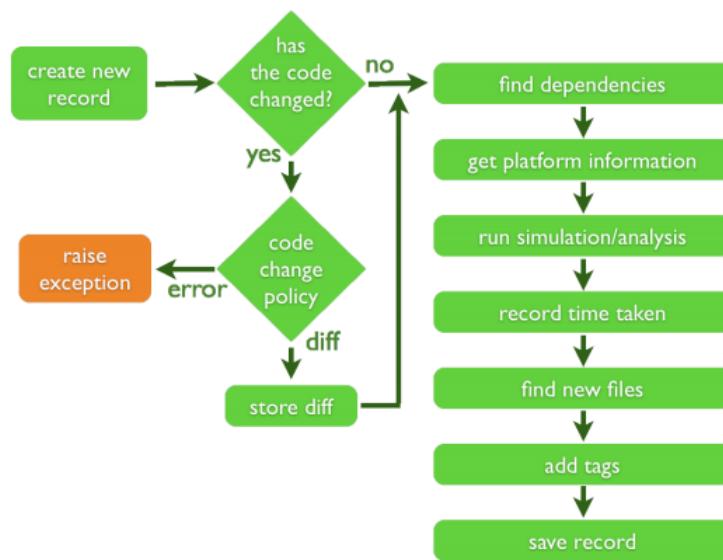


Figura 3.1: Passo-a-passo da coleta de dados do Sumatra [14].

Sumatra utiliza uma série de heurísticas para encontrar dependências para *scripts* Python, o que promove a reutilização de software e a reproduzibilidade [14]. Além disso, ele permite verificar e comparar experimentos que tiverem alguma alteração em seu contexto, porém, é necessário que o cientista utilize algum sistema de controle de versões para isso, como o Subversion, Git, Mercurial ou Bazar.

### 3.2.2 RDataTracker

RDataTracker é uma biblioteca de funções R que podem ser usadas para coletar proveniência de dados na forma de um DDG (*Data Derivation Graph*) durante sessões do console R ou durante a execução de um *script* R. Assim, o RDataTracker possui recursos para examinar e visualizar os dados de proveniência de *scripts* escritos em R, mantendo um alto nível de abstração para manter os grafos de proveniência comprehensíveis para os usuários [37].

Devido ao grande volume de dados em experimentos de larga escala, muitos *scripts* podem gerar grafos de proveniência realmente complexos de visualizar e interpretar, sendo

esta uma tarefa desafiadora para os cientistas que necessitam analisar a estrutura e os parâmetros de seus experimentos. Para resolver esse problema, o RDataTracker permite que o cientista introduza na ferramenta diferentes níveis de abstração para que seções inteiras dos grafos de proveniência possam ser compactadas em um único nó [37]. Isso é importante para evitar que *scripts* longos acabem resultando grafos de proveniência muito maiores do que o esperado.

O RDataTracker utiliza duas técnicas básicas para coletar a proveniência de *scripts* R [37]. Inicialmente, o cientista deve adicionar no *script* as chamadas relacionadas a biblioteca do RDataTracker, que são utilizadas para capturar a proveniência de dados durante a execução de um *script* R. Em segundo lugar, essas chamadas utilizam recursos do próprio R para examinar detalhes do *script*, tais como as chamadas de função e ligação entre as variáveis. Assim, um grafo de proveniência é gerado após a sessão de execução do *script*.

A Figura 3.2 mostra um exemplo de um grafo de proveniência construído pelo RDataTracker. Na Figura 3.2, os nós em amarelo representam uma declaração individual dos *scripts* R, tais como a atribuição de uma variável ou a chamada de uma função. Todos os nós na cor lilás representam os dados. A função *plot.data* (linha 14) por exemplo, recebe como parâmetro a variável *raw.data* (linha 13), que também é utilizada como parâmetro da função *calibrate* (linha 16).

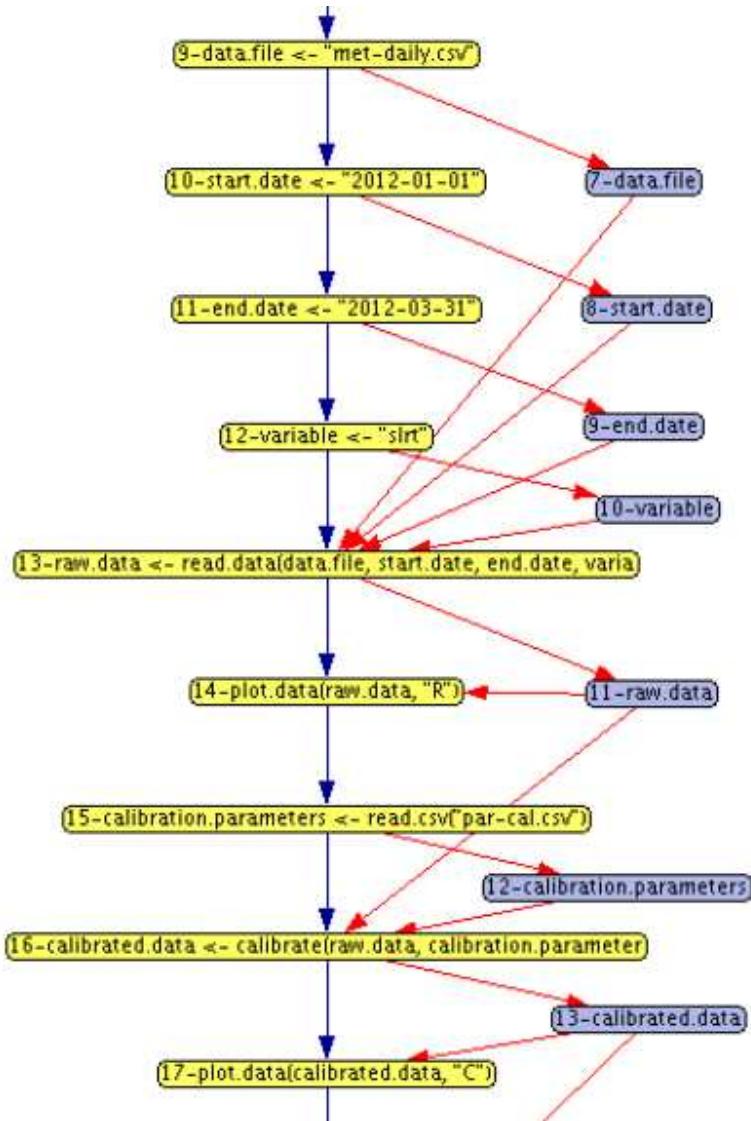


Figura 3.2: Grafo de proveniência gerado pelo RDATATracker [37].

### 3.2.3 ProvenanceCurious

O ProvenanceCurious é uma ferramenta que coleta a proveniência de *scripts* Python [32, 31]. Ele permite que os cientistas visualizem a proveniência de *scripts* Python por meio de grafos de proveniência. Dessa forma, ele possibilita mostrar valores que são gerados ao executar o *script*, como os valores de entrada e saída de uma função. A proposta do ProvenanceCurious é auxiliar os cientistas não só na interpretação de um *script*, mas também em visualizar as relações de dependência de dados entre as operações [32, 31].

O ProvenanceCurious permite que o usuário interaja diretamente com a ferramenta por meio de uma interface gráfica simples, recebendo como entrada um *script* em linguagem Python. Após a conclusão da geração do grafo, o ProvenanceCurious fornece várias opções para personalizar o grafo proveniência, aplicando regras para transformar o grafo

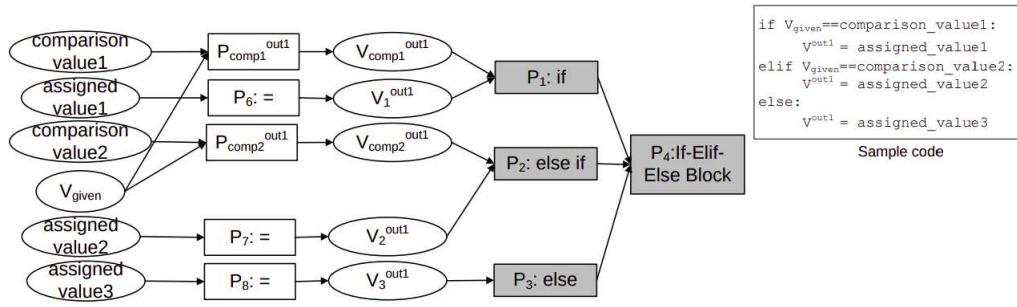


Figura 3.3: Grafo de proveniência do ProvenanceCurious [31].

de proveniência atual em uma versão personalizada [32, 31]. Para construir os grafos, primeiro, o ProvenanceCurious realiza uma análise de sintaxe nos dados recebidos por meio de uma *Abstract Syntax Tree (AST)* do *script*. Em seguida, os grafos de proveniência são gerados por meio do *AGG Engine (Graph Grammar Engine)*, um ambiente de desenvolvimento para sistemas de transformação de grafos, suportando uma abordagem algébrica na criação de modelos visuais em aplicativos JAVA.

A Figura 3.3 mostra um exemplo de grafo de proveniência prospectiva gerado pelo ProvenanceCurious. As setas representam o caminho do fluxo de dados e os vértices representam os blocos de código do *script*. O *P* representa um elemento de processamento, *V* denota um conjunto de nós que representam uma atividade. No diagrama, *P*<sub>1</sub>, *P*<sub>2</sub> e *P*<sub>3</sub> são os blocos condicionais (*if*, *elif* e *else*), *P*<sub>6</sub>, *P*<sub>7</sub> e *P*<sub>8</sub> indicam a atribuição das variáveis *V*<sup>out</sup>, enquanto *P*<sub>comp</sub> representa uma comparação dentro dos blocos de condição.

A Figura 3.4 apresenta detalhes da arquitetura do ProvenanceCurious. Primeiro, um *script* é fornecido como entrada pelo usuário. Esse *script* é analisado no módulo *Graph Building Engine*, responsável por realizar as análises e transformações relacionados ao *script* fornecido. O grafo de proveniência é criado no módulo *Graphical Interface* e os dados de proveniência coletados são armazenados e consultados no componente *Inference Engine*. O mecanismo *Inference Engine* é executado somente quando o usuário solicita depurar o modelo [32]. Além disso, os grafos de proveniência gerados pelo ProvenanceCurious podem ser customizados. Essa é a tarefa do módulo *Customization Engine*, que recebe as solicitações e opções de personalização por meio da interface gráfica da ferramenta.

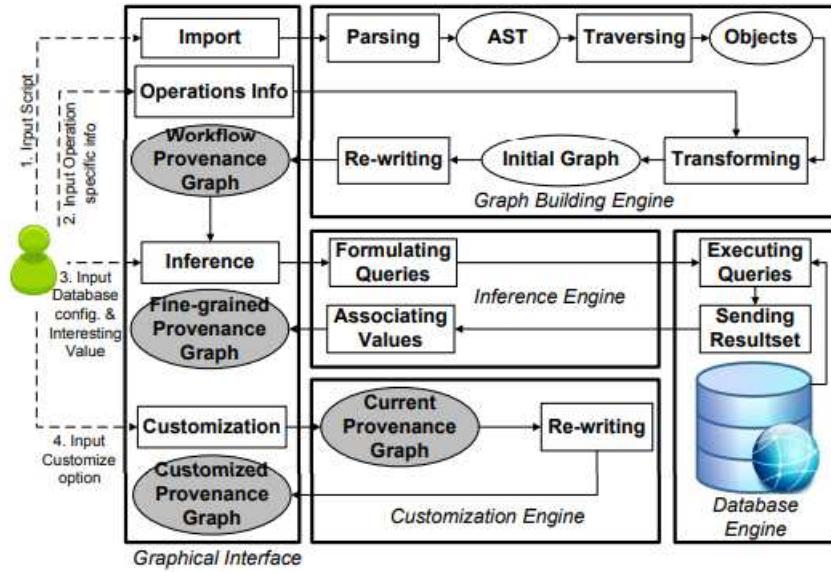


Figura 3.4: Arquitetura do ProvenanceCurious [32].

### 3.2.4 noWorkflow

O noWorkflow é uma ferramenta de código aberto que captura sistematicamente e de forma transparente a proveniência de *scripts* Python [44]. Durante a execução de um experimento em Python, o noWorkflow captura e armazena dados importantes de tal execução. Essas informações são utilizadas posteriormente para criar uma representação visual dos dados baseados na proveniência retrospectiva, isto é, as instruções executadas, incluindo também dados a respeito do ambiente computacional (bibliotecas instaladas, sistema operacional, informações de hardware, entre outros).

O noWorkflow permite a captura da proveniência de definição (*definition*), implantação (*deployment*) e execução (*execution*). Para a captura da proveniência de definição, correspondente a proveniência prospectiva, o noWorkflow usa uma AST (*Abstract Syntax Tree*). A AST é uma estrutura para representação sintática de códigos fonte [44, 53]. Com isso, o noWorkflow pode coletar informações a respeito do bloco de código de cada linha de um *script* Python, incluindo as estruturas de repetição e de condição, armazenando todos esses dados em um banco de dados SQLite.

Para coletar a proveniência de implantação, o noWorkflow usa bibliotecas fornecidas pelo próprio Python, extraíndo informações a respeito do ambiente de implantação, como versão do sistema operacional, variáveis de ambiente, arquitetura de hardware e bibliotecas instaladas [44, 53]. Tais informações são essenciais para identificar dependências e também

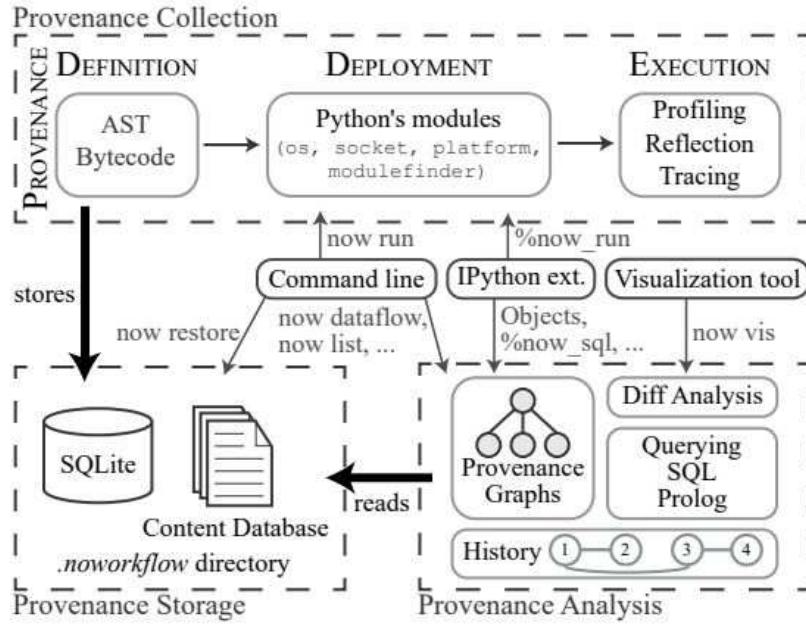


Figura 3.5: Arquitetura do noWorkflow [53].

para reproduzir o experimento em outra máquina.

Além dessas formas de coleta de proveniência, o noWorkflow também coleta a proveniência de execução, equivalente a proveniência retrospectiva. Com isso, o noWorkflow captura informações durante toda a execução de um *script* Python, o que inclui dados sobre ativações de função, valores retornados, atribuição de variáveis e o tempo de execução de cada uma das linhas executadas. Esses dados fornecem aos cientistas as informações necessárias para interpretar o que aconteceu em seu experimento ao longo do ciclo de execução, permitindo a compreensão das sequências envolvidas na produção do resultado final do experimento.

Além do que já foi mencionado, o noWorkflow permite coletar a proveniência em diferentes níveis de granularidade [44, 53, 50, 52]. Dessa maneira, para coletar a proveniência em granularidade grossa (*coarse-grained*), ele coleta dados sobre as ativações durante a execução do *script* e dados de entrada e saída (o que também inclui arquivos e variáveis). Já para a granularidade fina (*fine-grained*), o noWorkflow armazena as atribuições de variáveis e também as definições de *loop* [44, 53]. A Figura 3.5 mostra a arquitetura geral do noWorkflow, incluindo os métodos para captura de proveniência, estratégia de armazenamento e de análise.

O noWorkflow permite a visualização da proveniência por meio de grafos de dois tipos: grafos de ativação e *dataflows*. No grafo de ativação, as arestas representam o fluxo de informações, enquanto os vértices indicam as ativações. A Figura 3.6 mostra um grafo de

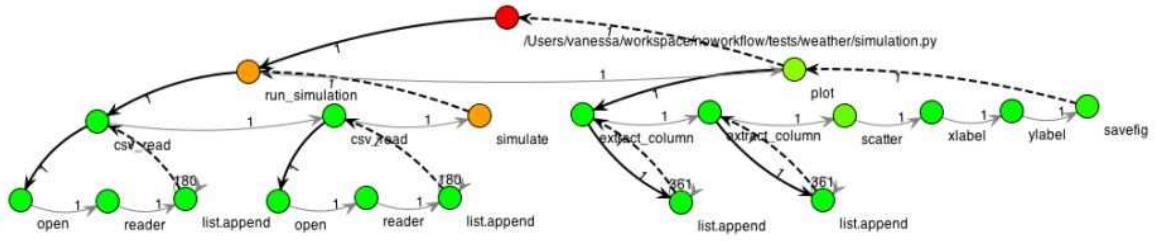
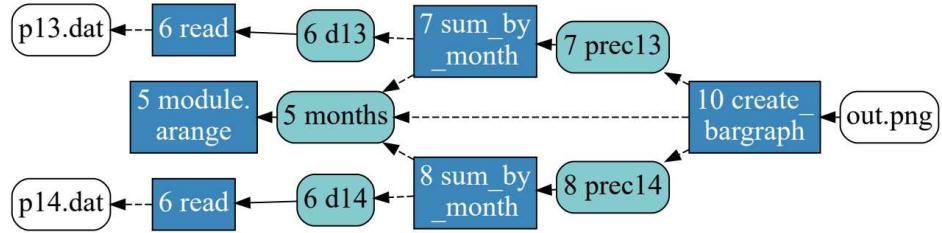


Figura 3.6: Grafo de ativação gerado pelo noWorkflow [44].

Figura 3.7: Dataflow gerado após execução do *script* de exemplo [53].

ativação gerado pelo noWorkflow. Cada nó é rotulado com o nome da função e a cor do nó segue a escala de um semáforo (a cor vermelha indica ativações mais lentas, amarelo indica ativações com tempo razoável e verde indica ativações mais rápidas). Em relação às arestas, as setas mais escuras são as chamadas de função, as tracejadas são os retornos e as setas em cinza claro são ativações sequenciais (quando duas funções são chamadas em sequência na mesma pilha de ativação).

Para evitar sobrecargas de visualização, o que pode ocorrer na presença de estruturas de repetição, o noWorkflow resume a proveniência antes de produzir o grafo de ativação [44, 53]. Para isso, ele realiza uma summarização, agregando ativações de uma chamada de função que pertencem a um mesmo *loop*. Depois disso, o noWorkflow realiza a construção do grafo a partir dos vértices gerados pela etapa de summarização e pelas arestas extraídas da sequência de ativação das funções executadas [44].

A Figura 3.7 mostra o *dataflow* gerado após a execução do *script* da Figura 3.8. Os retângulos em azul claro representam as variáveis, os retângulos em azul escuro representam as chamadas de função, os retângulos em branco representam os arquivos (*input* e *output*) e as arestas representam o fluxo de dados. O *script* da Figura 3.8 realiza a leitura de dados meteorológicos passados como parâmetro (*p13.data* e *p14.data*) e gera um gráfico (*out.png*) com a temperatura e precipitação na cidade do Rio de Janeiro em dois anos distintos (2013 e 2014).

```
01. import csv
02. import sys
03. import matplotlib.pyplot as plt
04. from simulator import simulate
05.
06. def run_simulation(data_a, data_b):
07.     a = csv_read(data_a)
08.     b = csv_read(data_b)
09.     data = simulate(a, b)
10.    return data
11.
12. def csv_read(f):
13.     reader = csv.reader(open(f, 'rU'), delimiter=',')
14.     data = []
15.     for row in reader:
16.         data.append(row)
17.     return data
18.
19. def extract_column(data, column):
20.     col_data = []
21.     for row in data:
22.         col_data.append(float(row[column]))
23.     return col_data
24.
25. def plot(data):
26.     # getting temperature
27.     t = extract_column(data, 0)
28.     # getting precipitation
29.     p = extract_column(data, 1)
30.     plt.scatter(t, p, marker='o')
31.     plt.xlabel('Temperature')
32.     plt.ylabel('Precipitation')
33.     plt.savefig('output.png')
34.
35. # main program
36. data_a = sys.argv[1]
37. data_b = sys.argv[2]
38. data = run_simulation(data_a, data_b)
39. plot(data)
```

Figura 3.8: Script de exemplo para gerar um gráfico de temperatura e precipitação durante um período de tempo específico [53].

Embora o noWorkflow colete a proveniência de definição, apenas a proveniência de execução é construída visualmente para representar dados de proveniência de um *script*. Mesmo não sendo utilizados para esse propósito no noWorkflow, a proveniência de definição (proveniência prospectiva) poderia ser utilizada para auxiliar os pesquisadores e desenvolvedores a interpretarem a estrutura de um *script* Python, sendo esse um dos principais objetivos dessa dissertação.

---

```

1 # @BEGIN collect_data_set
2 # @PARAM cassette_id @PARAM accepted_sample @PARAM num_images @PARAM energies
3 # @OUT sample_id @OUT energy @OUT frame_number
4 # @OUT raw_image_path @AS raw_image
5 # ... @URI file:run/raw/{cassette_id}/{sample_id}/e{energy}/image_{frame_number}.raw
6 run_log.write("Collecting data set for sample {}".format(accepted_sample))
7 sample_id = accepted_sample
8 for energy, frame_number, intensity, raw_image_path in collect_next_image(
9     cassette_id, sample_id, num_images, energies,
10    "run/raw/{cassette_id}/{sample_id}/e{energy}/image_{frame_number:03d}.raw"):
11    run_log.write("Collecting image {}".format(raw_image_path))
12 # @END collect_data_set
13
14 # @BEGIN transform_images
15 # @PARAM sample_id @PARAM energy @PARAM frame_number
16 # @IN raw_image_path @AS raw_image
17 # @IN calibration_image @URI file:calibration.img
18 # @OUT corrected_image @URI file:run/data/{sample_id}/{energy}eV_{frame_number}.img
19 # @OUT corrected_image_path @OUT total_intensity @OUT pixel_count
20    corrected_image_path = "run/data/{0}/{1}eV_{2:03d}.img".format(sample_id, energy, frame_number)
21    (total_intensity, pixel_count) = transform_image(raw_image_path, corrected_image_path, "calibration.img")
22    run_log.write("Wrote transformed image {}".format(corrected_image_path))
23 # @END transform_images

```

---

Figura 3.9: Exemplo de anotações na ferramenta YesWorkflow [42].

### 3.2.5 YesWorkflow

O YesWorkflow é um sistema que permite representar visualmente os dados coletados da proveniência prospectiva, sem necessidade de execução do código [42]. Para isso, o YesWorkflow utiliza um esquema de anotações que são incorporadas pelos cientistas ao código fonte do experimento. Tais anotações funcionam como orientação para o YesWorkflow gerar um grafo dirigido, onde cada vértice representa o bloco de código de uma determinada linha e as arestas simbolizam o fluxo de dados (entradas e saídas).

As anotações no YesWorkflow são baseadas em uma sintaxe própria, que indica o início e fim do código e outras informações relevantes para a construção do grafo. Além disso, o YesWorkflow suporta diferentes tipos de consultas para filtrar os dados, o que permite listar aspectos específicos da estrutura do *script*, como alguma chamada de função ou programa externo específico, parâmetros de entrada e saída de dados ou até blocos de código aninhados [42]. Como não realiza a execução do código fonte, o YesWorkflow processa as consultas levando em conta as anotações do usuário. Esse tipo de abordagem é muito vantajoso quanto a independência de sintaxe, já que é possível utilizar uma mesma anotação para diferentes linguagens de programação (o YesWorkflow suporta nativamente as linguagens Python, R e MATLAB). Em contrapartida, o modelo de anotações do YesWorkflow não garante a integridade das anotações, que podem se tornar obsoletas à medida em que o *script* evolui. A Figura 3.9 mostra um exemplo de anotações utilizando o YesWorkflow. O código em questão é delimitado pelas anotações *@BEGIN* e *@END*, que indicam o início e o final do que deve ser representado.

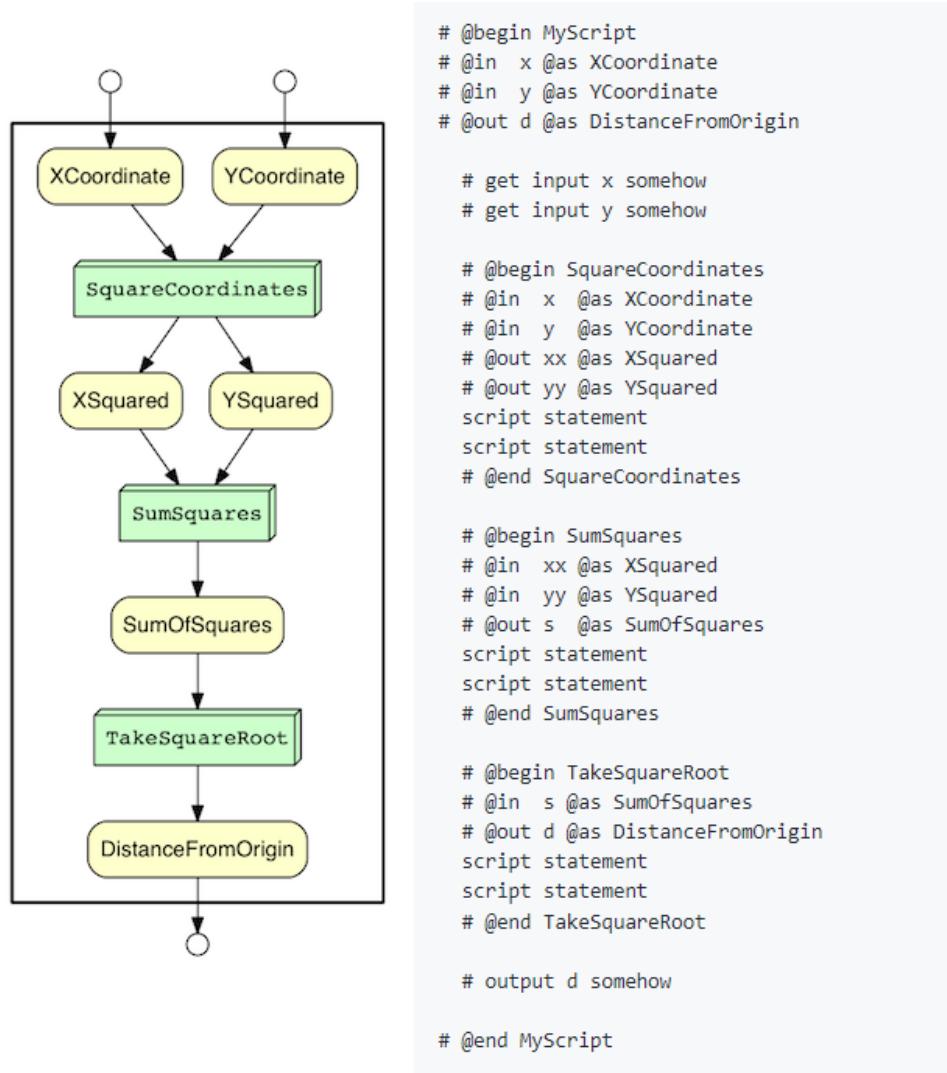


Figura 3.10: Grafo de proveniência gerado pelo YesWorkflow [42].

No YesWorkflow, a proveniência de definição é representada como um fluxo de processos. Os nós do grafo de proveniência representam os blocos de instrução e os dados declarados nas anotações, enquanto as arestas do grafo simbolizam a sequência das instruções e como elas se relacionam. A Figura 3.10 mostra um grafo de proveniência gerado com base em um *script* de exemplo para calcular a distância entre dois pontos (*XCoordinate* e *YCoordinate*). Além disso, o YesWorkflow também permite gerar dataflows. Nesse tipo de grafo de proveniência, os nós representam os dados (extraídos das anotações) e as arestas são anotadas com as instruções (também extraídas das anotações).

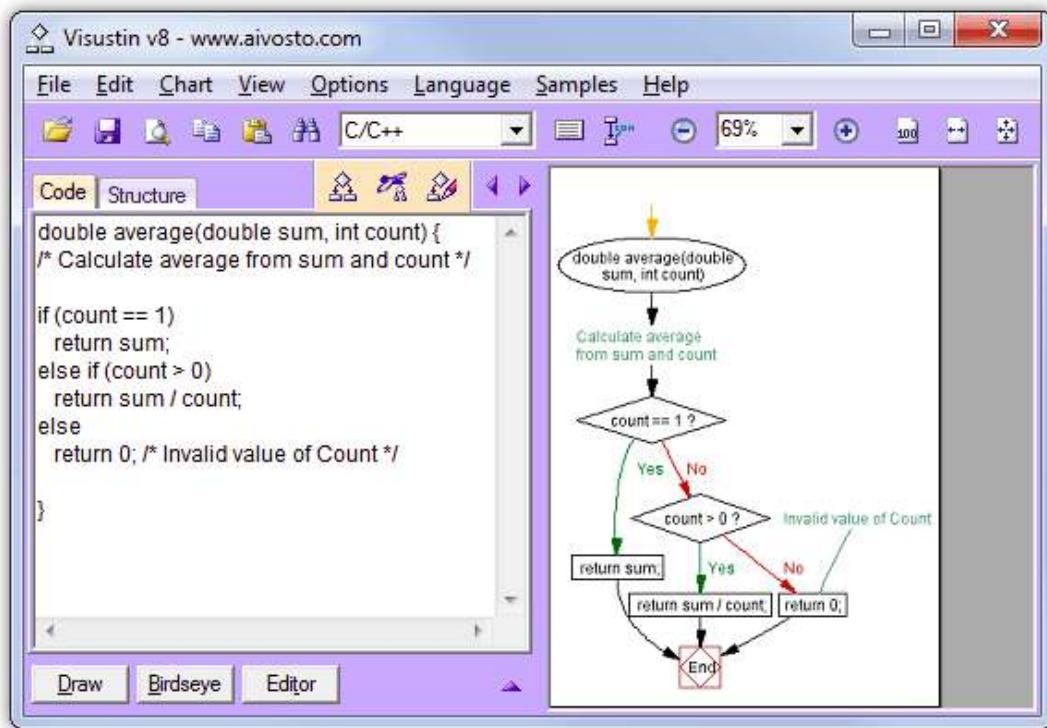


Figura 3.11: Interface para edição e criação de diagramas no Visustin [46].

### 3.2.6 Visustin

Visustin [46] é um sistema gerador de fluxogramas para desenvolvedores de software. Ele não se limita apenas à linguagem Python, gerando diagramas de fluxograma de outras linguagens de programação. Além disso, o Visustin converte automaticamente os códigos-fonte em diagramas de atividades UML. Os diagramas gerados pelo Visustin permitem visualizar a estrutura do código e os comentários. Embora seja uma ferramenta útil para representar código Python como diagramas, ela não permite verificar a corretude do código. De fato, o Visustin não executa os códigos e não garante que estejam sintaticamente corretos – os diagramas gerados baseiam-se apenas em regras básicas da linguagem e na indentação.

A Figura 3.11 mostra um exemplo de diagrama em linguagem C/C++ gerado pelo Visustin, incluindo sua interface gráfica para edição. No Visustin também é possível modificar o diagrama de forma interativa antes de exportá-lo. A mudança do diagrama ocorre à medida em que o usuário digita as modificações no campo de texto.

### 3.3 Ligação de Proveniência Prospectiva e Retrospectiva

Muitos dos sistemas mencionados anteriormente podem capturar tanto a proveniência de definição (ou prospectiva) quanto a proveniência de execução (ou retrospectiva). Embora a combinação da proveniência seja possível nesses sistemas, ela ainda é tratada de maneira independente, ou seja, as representações da proveniência são realizadas de forma isolada. Dito isso, nessa dissertação, também propomos a criação de uma visualização combinada entre os diferentes tipos de proveniências de *scripts* Python, que permite unir dados sobre a proveniência de definição e de execução em um mesmo modelo de representação.

Diante disso, é essencial mencionar o No+YesWorkflow [15], que propõe ligar dados da proveniência capturada pelo noWorkflow com o YesWorkflow. A integração entre proveniências apresentadas no trabalho de Dey et al. [15] mostra como as estruturas de fluxo produzidas a partir do YesWorkflow podem ser utilizadas para abstrair as informações coletadas pelo noWorkflow. O No+YesWorkflow armazena a proveniência prospectiva em conformidade com D-PROV, além de guardar o mapeamento entre ambas as proveniências (prospectiva e retrospectiva), permitindo também que os usuários consultem interativamente os dados de proveniência coletados [15].

A Figura 3.12 mostra a arquitetura do No+YesWorkflow. No componente *Prospective Provenance Generator*, o No+YesWorkflow gera proveniência prospectiva, usando para isso dados capturados pelo YesWorkflow. Já a proveniência retrospectiva é gerada no componente *Retrospective Provenance Generator*, que usa dados de proveniência coletados diretamente pela ferramenta noWorkflow após a execução do *script*. Por fim, o componente *Provenance Integrator* mapeia e integra as proveniências prospectiva e retrospectiva.

Embora o No+YesWorkflow possa combinar a proveniência prospectiva com a retrospectiva, ainda existe um problema na forma em que isso é feito, pois a união de proveniências no No+YesWorkflow é totalmente dependente das anotações que o usuário realiza no *script* Python. Dessa forma, o No+YesWorkflow não permite executar esse processo de maneira automática: tanto a anotação do *script*, usando a notação do YesWorkflow, quanto o mapeamento entre as anotações e a proveniência retrospectiva gerada pelo noWorkflow precisam ser feitas manualmente, o que exige um grande esforço por parte do usuário.

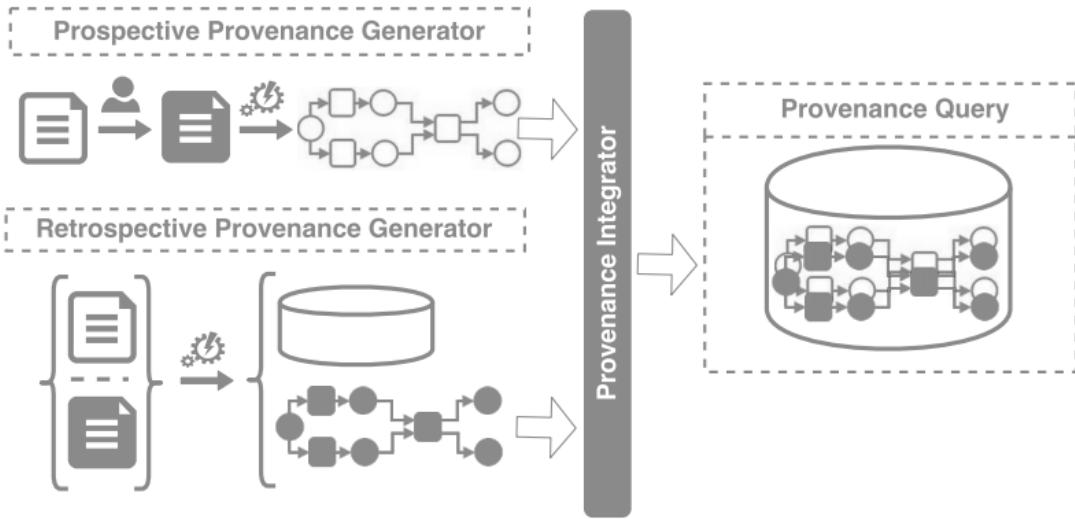


Figura 3.12: Arquitetura do No+YesWorkflow [15].

## 3.4 Considerações Finais

Muitos sistemas abordam a coleta de proveniência em diferentes cenários. Nesse capítulo apresentamos detalhes sobre sistemas que capturam diferentes tipos de proveniência em *scripts*. Dentre esses sistemas, destacamos o noWorkflow [53], ProvenanceCurious [31, 32], YesWorkflow [42] e No+YesWorkflow [15]. Tais abordagens são muito relevantes para essa dissertação, pois possibilitam a captura de proveniência em *scripts* Python, e geram diagramas para representá-la. Além disso, também devemos mencionar o Visustin [46], que embora não seja uma abordagem para a coleta de proveniência, permite gerar diagramas de *scripts*.

A Tabela 3.1 apresenta mais informações a respeito de diversos sistemas de captura de proveniência, inclusive detalhes sobre a forma de coleta, granularidade e propósito geral de cada um deles. Cabe notar que, das ferramentas que suportam proveniência de *scripts*, apenas o No+YesWorkflow permite representar proveniência prospectiva e retrospectiva em diferentes níveis de granularidade, com a ressalva de que o mapeamento precisa ser feito de forma manual. Dessa forma, acreditamos que há espaço na literatura para uma abordagem que consiga conciliar esses tipos de proveniência de forma automática. É importante ressaltar que o Visustin não está inserido nesta tabela, pois ele não é um sistema para captura de proveniência. Além disso, a tabela também apresenta abordagens de captura de proveniência de *workflows* e de dados, embora não mencionadas neste capítulo, pois não permitem capturar a proveniência de *scripts*.

Tabela 3.1: Sistemas para Captura de Proveniência.

Sistemas	Citação	Propósito Geral	Tipo		Granularidade	
			Prospect.	Retrospect.	Grossa	Fina
Galaxy	[27]	<i>Workflows</i>	Não	Sim	Sim	Sim
Taverna	[30]		Sim	Sim	Sim	Sim
VisTrails	[10]		Não	Sim	Sim	Não
Kepler	[5]		Sim	Sim	Sim	Sim
YesWorkflow	[42]	<i>Scripts</i>	Sim	Não	Sim	Não
noWorkflow	[53, 44]		Não	Sim	Sim	Sim
No+Yesworkflow	[15]		Sim	Sim	Sim	Sim
ProvenanceCurious	[32, 31]		Sim	Não	Não	Sim
RDataTracker	[37]		Não	Sim	Sim	Não
HadoopPROV	[4]	<i>Data Provenance</i>	Não	Sim	Sim	Sim
Titian	[33]		Não	Sim	Sim	Sim
Ariadne	[26]		Não	Sim	Sim	Sim

# Capítulo 4

## ProspectiveProv

### 4.1 Introdução

Em ensaios científicos baseados em *scripts*, entender a estrutura do experimento é essencial, principalmente para a reprodução, modificação e revisão do experimento. Além disso, compreender o comportamento de cada parte do código durante a execução ajuda os usuários a identificarem falhas graves e possíveis inconsistências no *script*, auxiliando também na otimização dos códigos e na análise de resultados. Para facilitar como o cientista vê e interpreta o seu experimento, desenvolvemos o ProspectiveProv, uma ferramenta que permite representar os dados de proveniência por meio de grafos baseados na proveniência prospectiva de um *script*.

O ProspectiveProv é um mecanismo que utiliza dados de proveniência para gerar grafos de proveniência prospectiva de *scripts* Python. Muitos detalhes podem passar despercebidos ao olhar do usuário que está analisando um *script*. É nesse sentido que o ProspectiveProv atua, gerando diagramas como forma de representar visualmente um *script* de maneira simples, indicando a relação existente entre cada bloco de código.

Os detalhes sobre o ProspectiveProv são abordados neste capítulo, que está organizado como segue. Na Seção 4.2, a arquitetura do ProspectiveProv é apresentada. Na Seção 4.3, discutimos como a ferramenta acessa e organiza os dados de proveniência. Na Seção 4.4, mostramos detalhes sobre como o ProspectiveProv mapeia cada componente. Na Seção 4.5, mostramos como os grafos de proveniência prospectiva são gerados na ferramenta. Na Seção 4.6 apresentamos como os grafos de proveniência prospectiva são combinados a dados de proveniência retrospectiva. A Seção 4.7 explica como o ProspectiveProv pode ser usado na prática. Por fim, a Seção 4.8 apresenta as considerações finais deste capítulo.

## 4.2 Arquitetura

Como visto em capítulos anteriores, proveniência prospectiva refere-se à estrutura do experimento [53]. Para construir os grafos com base em dados de proveniência, o ProspectiveProv funciona de maneira automática. Antes de mais nada, os experimentos de *script* devem ser executados pelo noWorkflow [44, 53]. Após esse processo, o usuário informa ao ProspectiveProv a identificação do *trial* do experimento que foi executado anteriormente no noWorkflow, não sendo necessário inserir informações manualmente no *script* como ocorre no esquema de anotações no YesWorkflow. Além disso, o ProspectiveProv possibilita combinar a proveniência prospectiva e retrospectiva, utilizando para isso, um mecanismo baseado em *hash* para identificar cada nó do grafo do *script* e garantir a ligação entre eles.

O ProspectiveProv é composto por quatro módulos: *Experiment data collector*, *Hash mapper*, *Graph builder* e *Graph drawer*. Primeiro, para utilizar o ProspectiveProv, o *script* Python precisa ser executado usando a interface de comando do noWorkflow, que se encarregará de capturar e armazenar localmente os dados de proveniência. Essa etapa é necessária pois, como o ProspectiveProv não captura dados de proveniência, ele utiliza o noWorkflow como mediador, conectando-se diretamente à base de dados SQLite do noWorkflow para acessar dados relacionados à proveniência capturada durante a execução de um experimento.

A Figura 4.1 mostra os componentes da arquitetura do ProspectiveProv. Primeiro, o módulo *Experiment Data Collector* se conecta à base de dados SQLite do noWorkflow. São realizadas consultas SQL para acessar os dados de proveniência armazenados na base SQLite. Tais dados são armazenados localmente em forma de um *array list* em memória principal. Depois desse processo, os dados extraídos são enviados ao *Hash Mapper*, módulo responsável por criar endereços *hash* para cada nó do grafo de *script*. Após mapear todos os componentes, o *Graph Builder* entra em ação, analisando a sintaxe do código Python e estabelecendo regras de como os nós devem ser conectados (considerando a simbologia dos *flowcharts* e regras da linguagem Python 3). O componente *Graph Builder* não funciona sozinho, ele é integrado ao *Graph Drawer*, módulo responsável por desenhar os grafos de proveniência usando o Graphviz, um pacote de ferramentas *open-source* para desenhar gráficos em linguagem DOT.

O ProspectiveProv produz grafos seguindo fortemente a simbologia dos famosos *flowcharts*, um tipo de diagrama muito utilizado para descrever processos e algoritmos. Dessa

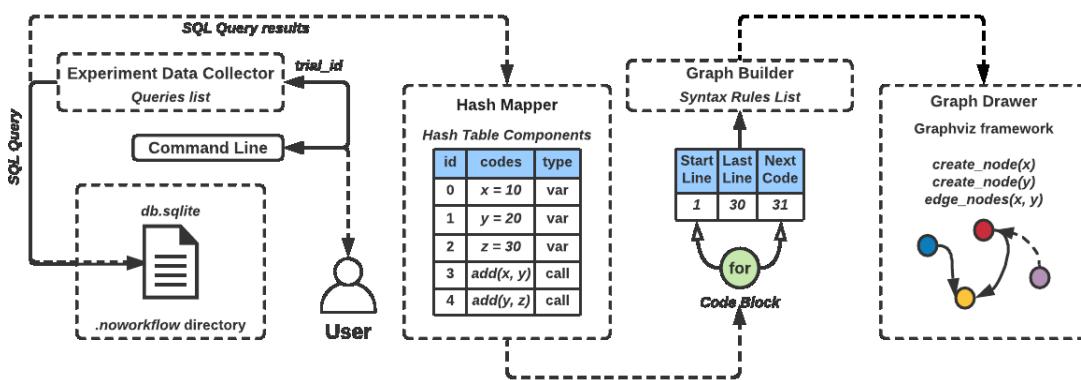


Figura 4.1: Arquitetura do ProspectiveProv.

forma, o ProspectiveProv foge do modelo tradicional da maioria dos sistemas que possibilitam coletar e representar a proveniência de *scripts*. Utilizando os *flowcharts*, é possível representar com clareza a estrutura do *script*, isto é, a maneira como a sequência de passos do *script* está organizada. Optamos por representar a proveniência dessa forma devido ao alto nível de abstração dos fluxogramas.

## 4.3 Acesso a Dados de Proveniência

ProspectiveProv coleta dados de proveniência armazenados pelo noWorkflow. Para armazenar os dados de proveniência, o noWorkflow utiliza um banco de dados SQLite, que é armazenado dentro do diretório `.noworkflow` na pasta do *script*. Na base de dados, ele armazena a proveniência prospectiva na tabela `code_component`, que contém dados relacionados à estrutura do *script*. No caso da proveniência retrospectiva, é possível obter os dados de variáveis dentro da tabela `evaluation`.

A Figura 4.2 mostra detalhes importantes sobre o modelo de dados do noWorkflow. O ProspectiveProv coleta dados da tabela `trial` para verificar o status do experimento, `code_component` para obter dados dos *script* e `evaluation` para obter informações de funções ativadas e o conteúdo das variáveis coletados durante a execução do código.

Antes de requisitar qualquer acesso aos dados, o ProspectiveProv verifica se o diretório `.noworkflow` existe e se o experimento tem uma execução bem-sucedida (verificando o status da `trial` consultada). Em seguida, é estabelecida uma conexão direta com a base de dados SQLite do noWorkflow. Tanto o acesso à proveniência quanto a extração de dados são feitas pelo módulo *Experiment Data Collector*.

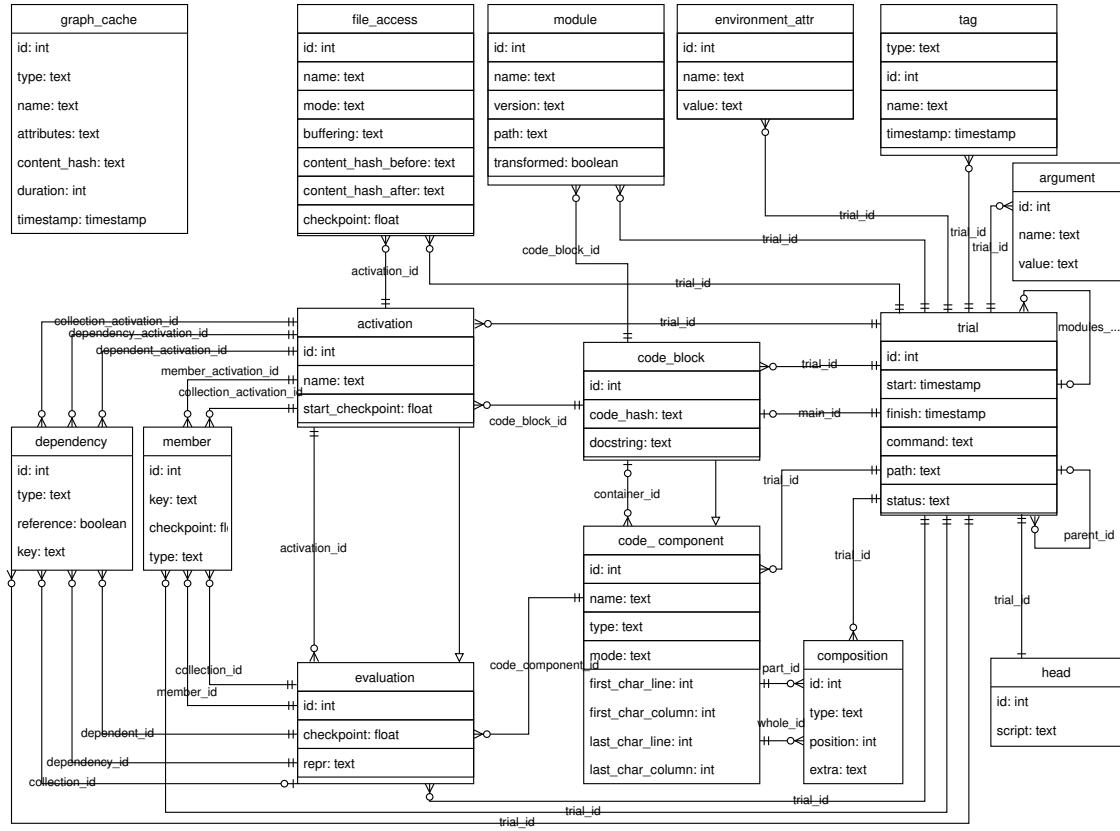


Figura 4.2: Modelo de dados do noWorkflow [49]

A Figura 4.3 mostra a lista de consultas SQL utilizadas pelo ProspectiveProv para obter dados de proveniência prospectiva e retrospectiva coletadas pelo noWorkflow. A Figura 4.4 mostra detalhadamente o funcionamento do módulo *Experiment Data Collector*. Quando o usuário insere na interface de comando do ProspectiveProv o parâmetro identificador (*trial id* do experimento), tal parâmetro é inserido na consulta. A consulta é então enviada para o *SQLite Database Connection*, função contendo as chamadas necessárias para acessar a base de dados SQLite usando o Python. Depois do processo de consulta, os dados de proveniência retornados são armazenados localmente em uma *array list*, que será utilizada posteriormente para gerar os grafos de proveniência prospectiva.

## 4.4 Mapeamento dos Nós do Grafo

O mapeamento dos componentes é necessário para garantir que todos os nós estejam ligados corretamente. Diante disso, após a consulta dos dados de proveniência (visto na Seção 4.3), o ProspectiveProv mapeia cada componente. Assim, cada nó do grafo recebe um identificador único, formado pela linha inicial do código, o tipo e a coluna em que ele se encontra. Além disso, usando esse esquema, é possível acessar um nó específico do

```

1
2  -- LISTA DE CONSULTAS - PROSPECTIVE PROV
3  -- Verificar se o experimento foi executado corretamente
4  SELECT status FROM trial WHERE id = trial;
5
6  -- Obter dados do código - proveniencia prospectiva
7  SELECT first_char_line, last_char_line FROM code_component WHERE trial_id = trial;
8
9  -- Obter os blocos que foram ativados - proveniencia retrospectiva
10 SELECT code_component.first_char_line, evaluation.checkpoint FROM evaluation
11   JOIN code_component WHERE evaluation.code_component_id = code_component.id AND
12     evaluation.trial_id = code_component.trial_id AND
13     evaluation.trial_id = trial AND code_component.first_char_line != -1
14   ORDER BY code_component.first_char_line;
15
16  -- Obter os dados de execução - proveniencia retrospectiva
17  SELECT code_component.first_char_line, evaluation.checkpoint FROM evaluation
18    JOIN code_component WHERE evaluation.code_component_id = code_component.id AND
19      evaluation.trial_id = code_component.trial_id AND
20        evaluation.trial_id = trial AND
21          code_component.first_char_line != -1 AND
22            code_component.type == 'name'
23  ORDER BY code_component.first_char_line;
24
25  -- Obter os dados de execução - proveniencia retrospectiva
26  SELECT code_component.first_char_line, evaluation.repr FROM evaluation
27    JOIN code_component WHERE evaluation.code_component_id = code_component.id AND
28      evaluation.trial_id = code_component.trial_id AND evaluation.trial_id = trial AND
29        code_component.first_char_line != -1 AND
30          code_component.type == 'name'
31  ORDER BY code_component.first_char_line;
32

```

Figura 4.3: Módulo de coleta de dados de proveniência.

grafo a qualquer momento, permitindo ligar outros componentes ou alterar a configuração dele (nome, cor, formato).

Para exemplificar isso, a Figura 4.5 e a Figura 4.6 mostram como o ProspectiveProv cria o identificador para os nós do grafo de proveniência. Na Figura 4.5, uma tabela é recebida como entrada para a função criar  $hash(x)$ . Dessa tabela, são extraídos a linha inicial, a categoria e a coluna em que o código se encontra. Assim, a função criar  $hash(x)$  sumariza esses itens em uma única *string*. A Figura 4.6 mostra a segunda etapa desse algoritmo. Na segunda etapa, o módulo do Graphviz [21] recebe o *hash* do nó que ele deve criar. O formato do nó é determinado com base no tipo de componente de código, levando em consideração as regras de simbologia dos fluxogramas de algoritmo. Finalmente, depois de criar os nós e interliga-los no grafo do *script*, o ProspectiveProv gera um arquivo em PDF.

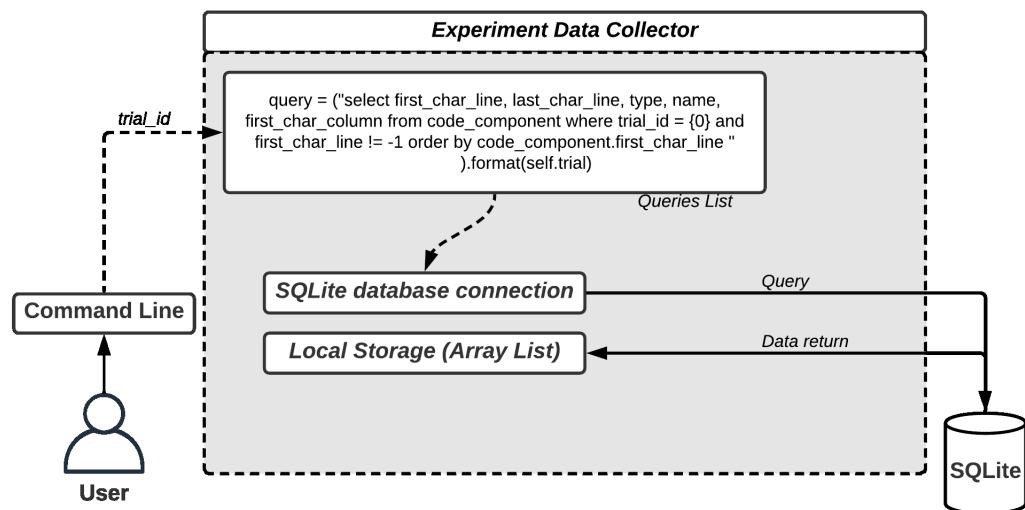


Figura 4.4: Lista de consultas utilizadas pelo ProspectiveProv para consultar os dados de proveniência.

## 4.5 Visualização da Proveniência

Definir um algoritmo para construir grafos de proveniência é complicado devido ao grande número de detalhes para gerenciar [3]. Além disso, é necessário trabalhar em métodos que garantam a eficiência nesse processo, pois um grafo de proveniência deve transmitir as informações obtidas de forma concisa e de fácil compreensão.

Em uma primeira abordagem, abandonamos o modelo tradicional baseado em notações de proveniência. Ao invés disso, o ProspectiveProv produz grafos baseados no conceito de fluxogramas, como mostrado anteriormente. De acordo com Scanlan [60], comparado a pseudocódigos, os fluxogramas estruturados ajudam os usuários a analisar mais rapidamente os algoritmos, mesmo quando se trata de iniciantes com pouco contato com linguagens de programação. Os fluxogramas auxiliam na interpretação da estrutura de um código. Muitos outros trabalhos na literatura demonstram as vantagens da utilização desse tipo de diagrama para representar algoritmos [13, 17, 60, 65].

O ProspectiveProv trabalha com *script* Python que utilizam construções simples. Nesse ponto, ele ainda possui deficiência em representar estruturas de *scripts* que utilizam conceitos mais complexos de orientação a objetos e funções *lambda*. Embora seja uma limitação, os cientistas, em sua maioria, utilizam construções mais simples em seus experimentos de *script*, tais como atribuição de variáveis, importações, *loops*, e estruturas condicionais [49]. A Figura 4.7 mostra a simbologia utilizada nos diagramas do ProspectiveProv para representação da proveniência prospectiva e retrospectiva de *scripts*.

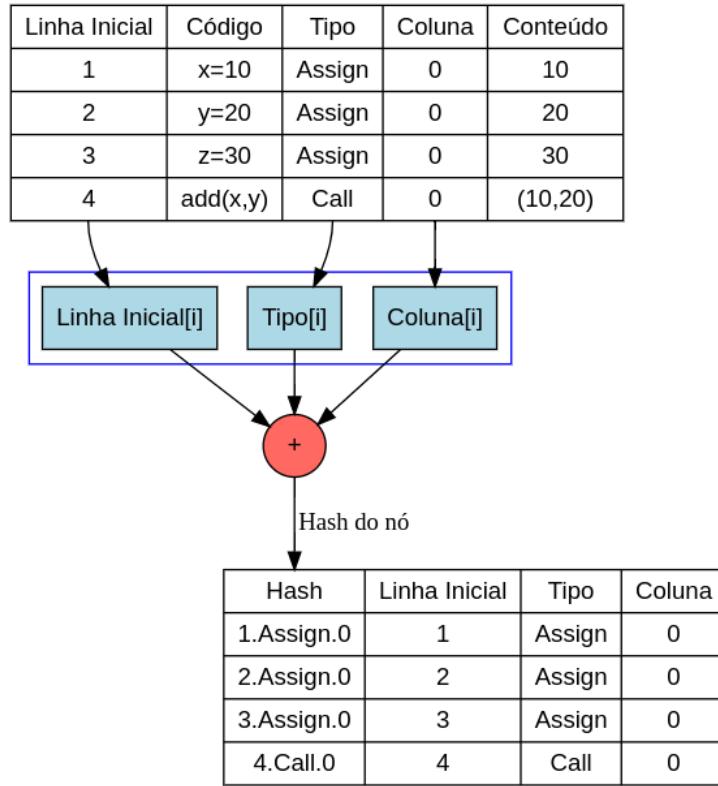


Figura 4.5: Rotulação dos nós do grafo de proveniência.

Python.

A Figura 4.8 mostra um experimento simples, cujo grafo foi construído usando as notações de fluxogramas definidas pelo ProspectiveProv. Na Figura 4.8, os retângulos em cor verde indicam as chamadas de função, a elipse em verde indica um *loop*, os retângulos rosas representam a atribuição de uma variável, os blocos de anotação em branco indicam as condições presentes no *loop* e na estrutura condicional. As setas pretas representam o fluxo de controle do *script*, enquanto as setas pretas pontilhadas indicam o fluxo de retorno de um laço de repetição.

A Figura 4.9 mostra um diagrama gerado pelo ProspectiveProv com a união de proveniências prospectiva e retrospectiva. Na Figura 4.9, os losangos em cor verde indicam laços condicionais, a elipse em verde indica um *loop*, os retângulos rosas representam a atribuição de uma variável, os retângulos cinzas indicam os blocos de código que não foram ativados durante a execução, os blocos de anotação em branco indicam a condição do laço condicional e do *loop*, os blocos de anotação em amarelo indicam os valores que variável recebeu durante a execução.

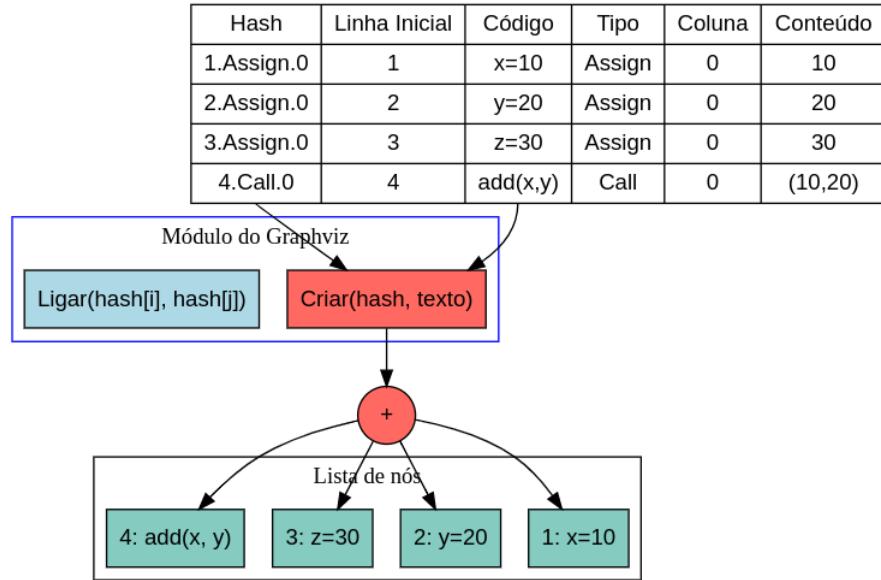


Figura 4.6: Gerando e ligando novos nós do grafo de proveniência.

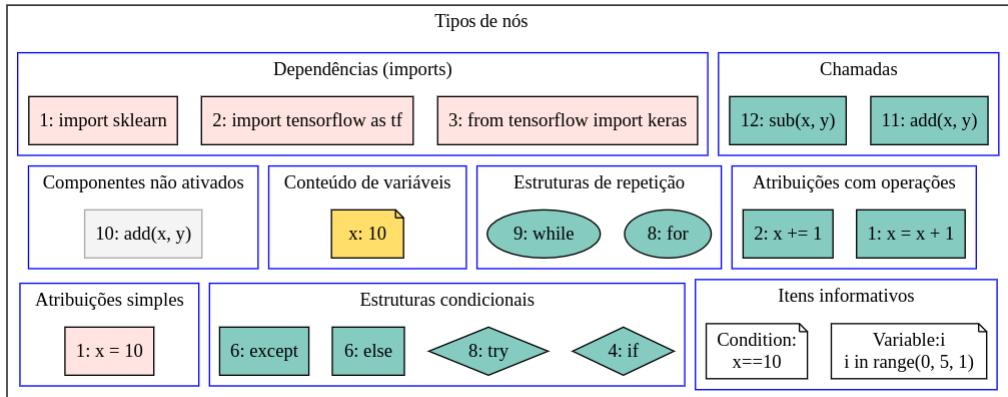


Figura 4.7: Simbologia utilizada para representar os nós no ProspectiveProv.

## 4.6 Conectando Proveniência Prospectiva e Retrospectiva

Os dados referentes a proveniência retrospectiva podem incluir detalhes relevantes para análise de um experimento de *script*, tais como o tempo de execução de cada bloco de código, o conteúdo das variáveis em um determinado instante e informações sobre o ambiente computacional. Em experimentos de *script*, exibir o conteúdo das variáveis a cada instante pode ser importante para o cientista entender como os parâmetros inseridos estão se comportando durante a execução do código. Para ajudar os cientistas nessa tarefa, ferramentas como o noWorkflow [53, 44] permitem coletar e representar a proveniência retrospectiva de ensaios de *script*. Nessa dissertação, combinamos dados de ambos os

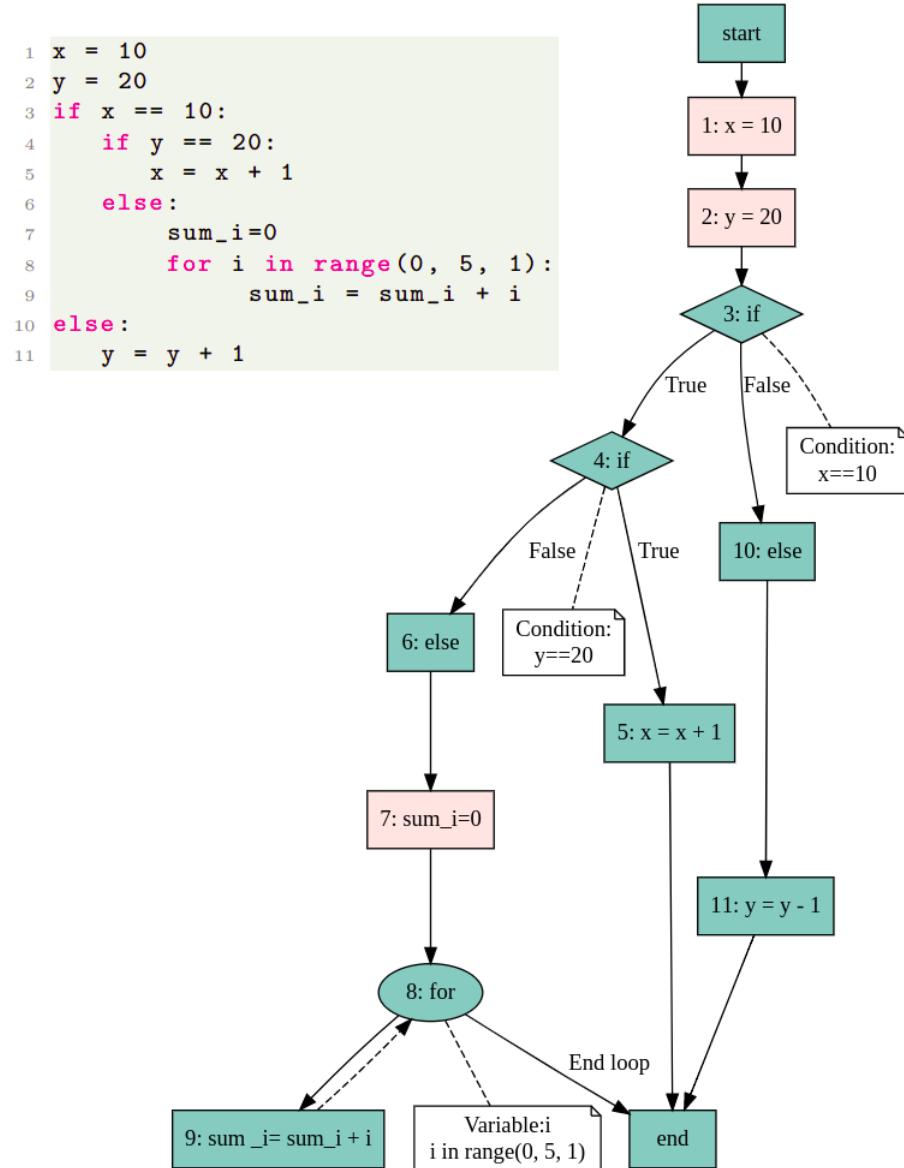


Figura 4.8: Diagrama de proveniência prospectiva produzido pelo ProspectiveProv.

tipos de proveniência (prospectiva e retrospectiva), possibilitando que os pesquisadores tenham dados da estrutura de seu experimento e ao mesmo tempo visualizem dados sobre a execução do ensaio (como *logs* de tempo, conteúdo de variáveis e funções ativadas).

Combinar dados entre diferentes tipos de proveniência não é inédito. Nesse quesito, o No+Yesworkflow [42] apresenta uma proposta similar, porém, ainda é bastante limitada, já que depende de anotações que o usuário realiza no *script* do experimento. Diante disso, uma das propostas do ProspectiveProv é contornar esse problema, permitindo que os grafos sejam produzidos de forma automática, sem a necessidade de criar anotações no ensaio ou usar outro mecanismo manual.

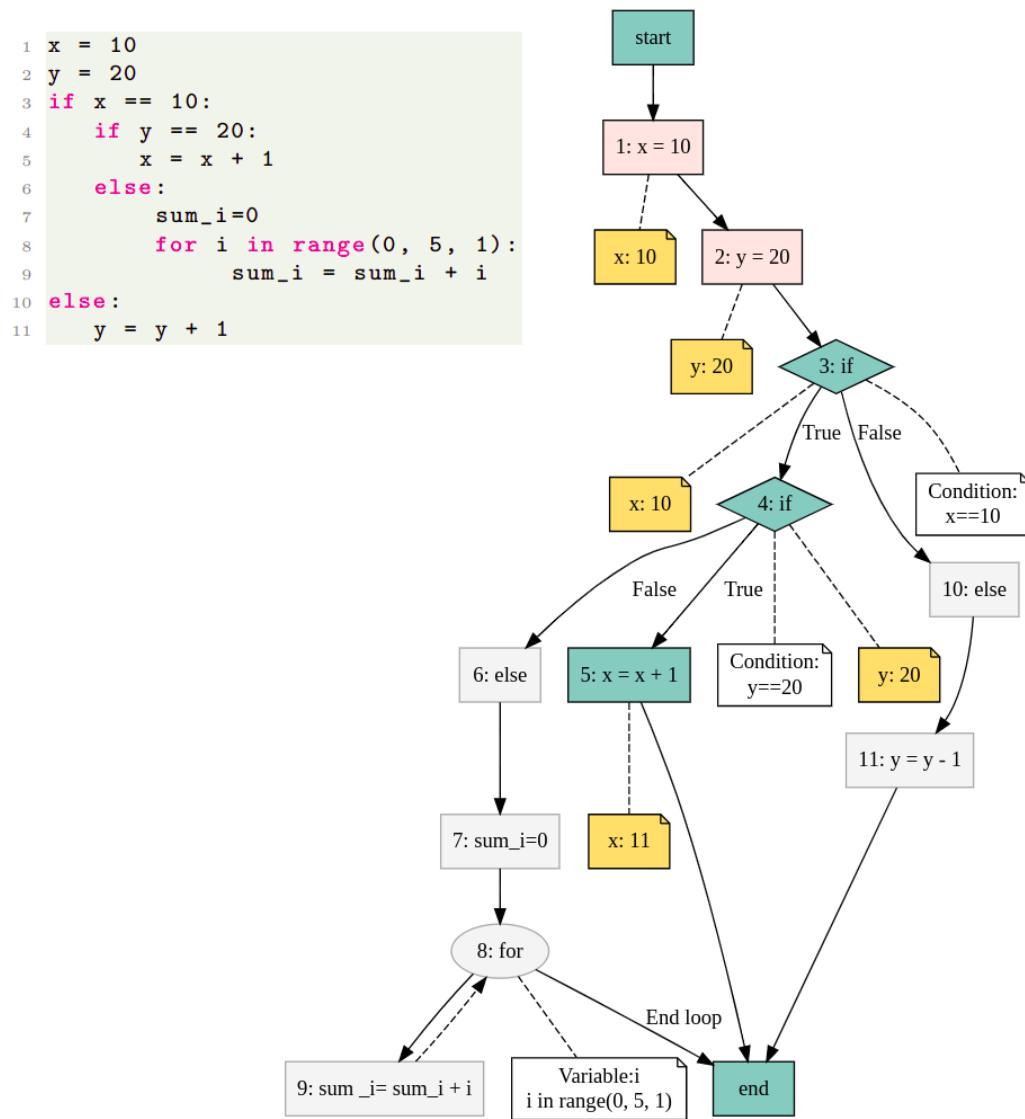


Figura 4.9: Diagrama com a união da proveniência prospectiva e retrospectiva produzido pelo ProspectiveProv.

Para ligar ambos os tipos de proveniência no ProspectiveProv, é necessário vincular o nó do grafo do *script* com os dados de execução coletados pelo noWorkflow, como o conteúdo das variáveis e as chamadas ativadas. A Figura 4.10 mostra detalhes de como a proveniência retrospectiva é ligada ao grafo de proveniência prospectiva no ProspectiveProv. Nesse exemplo, os nós do grafo já foram previamente criados e este processo é realizado após a finalização da geração do grafo de *script*. Para representar o conteúdo das variáveis, o ProspectiveProv cria caixas em cor amarela. Cada caixa é vinculada diretamente ao nó que representa uma determinada linha do código. Para que essa ligação seja possível, os endereços hash são reconstruídos, gerando assim o endereço do nó em que as caixas amarelas devem ser combinadas.

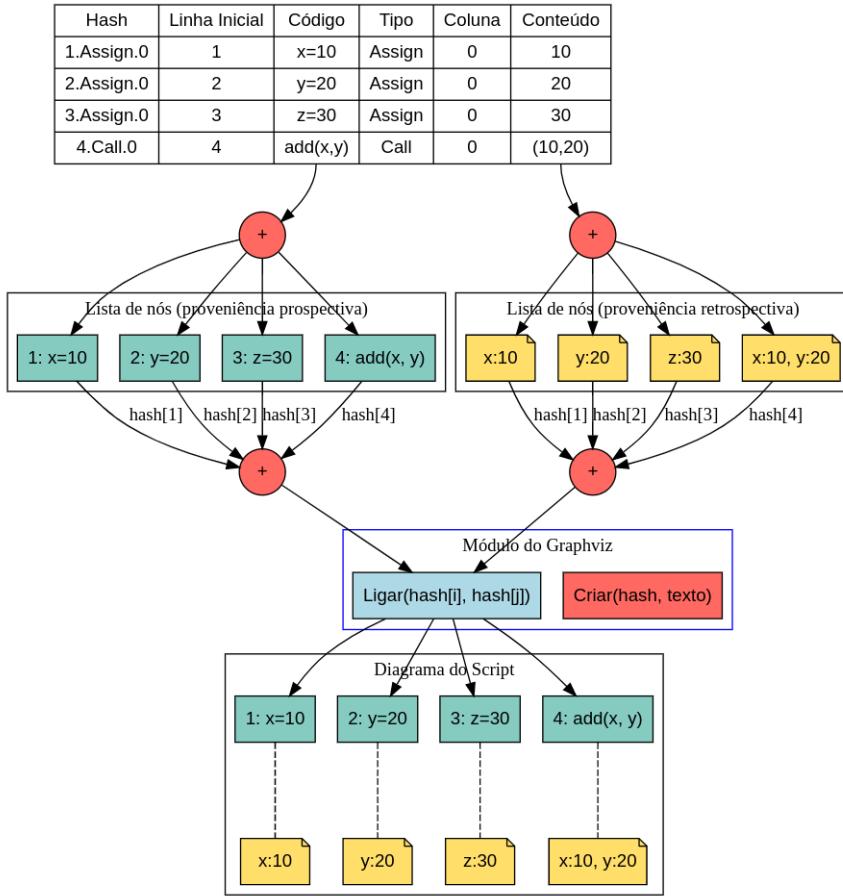


Figura 4.10: Ligando dados de proveniência retrospectiva e prospectiva.

## 4.7 Uso do ProspectiveProv

Para usar o ProspectiveProv, é necessário executar o *script* do experimento Python usando o noWorkflow. Cada experimento executado no noWorkflow é armazenado usando uma identificação numérica, chamada de *trial* [53]. Após a execução do *script*, é possível executar o ProspectiveProv e gerar diagramas de proveniência prospectiva utilizando o comando `python3 prospective.py --trial <trial_id>`, onde `<trial_id>` refere-se a *trial* do experimento no noWorkflow. Tal comando permite exibir no diagrama dados referentes à estrutura do *script*, tais como as chamadas, funções, condições, *loops*, variáveis e classes. A Figura 4.8 mostra um exemplo de grafo gerado por esse comando.

Quanto à combinação entre as proveniências prospectiva e retrospectiva, é possível utilizar o ProspectiveProv para gerar diagramas com o conteúdo das variáveis *script* num determinado *trial*. Para ativar este modo, é necessário executar o comando `python3 prospective.py --trial <trial_id> --var 1` (o valor 1 indica que esta função está ativada). Também é possível exibir os blocos de código que foram ativados e não ativados

durante o ciclo de execução do experimento. Este modo é ativado utilizando o comando `python3 prospective.py --trial <trial_id> --act 1`. Ambos os modos são representados junto do diagrama de proveniência prospectiva, o que permite compreender a estrutura do código e detalhes de sua execução. Além disso, é possível combinar modos de visualização, utilizando o comando `python3 prospective.py --trial <trial_id> --var 1 --act 1` por exemplo, permitindo neste caso gerar um diagrama com o conteúdo das variáveis e os códigos não ativados. A Figura 4.9 mostra um exemplo de grafo com a combinação da proveniência prospectiva e a retrospectiva (ativações e conteúdo das variáveis).

Para *scripts* com muitas linhas, o que pode gerar grafos com muitos nós, é possível utilizar o ProspectiveProv para exibir apenas o diagrama de um subconjunto de linhas do *script*. Assim, é necessário usar a opção `--show l <11> <12>`, onde `<11>` indica a linha inicial e `<12>` indica a linha final. Para exibir apenas uma função específica, pode-se utilizar a opção `--show f <fname>`, onde `<fname>` refere-se ao nome da função dentro do código Python.

A seguir disponibilizamos um resumo dos comandos possíveis do ProspectiveProv. Para esse exemplo, executamos um experimento de um *script* Python cujo valor da *trial* é 1:

- `python3 prospective.py --trial 1`: gera um diagrama de proveniência prospectiva (ver Figura 4.11).

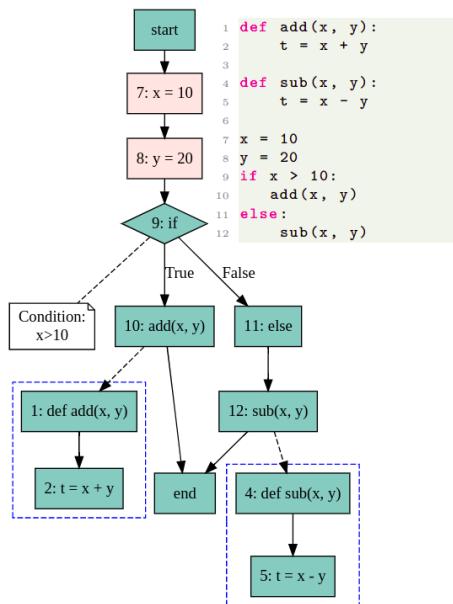


Figura 4.11: Diagrama de proveniência prospectiva.

- `python3 prospective.py --trial 1 --i 1`: gera um diagrama de proveniência prospectiva e também exibe a endentação em cada bloco de código (ver Figura 4.12).

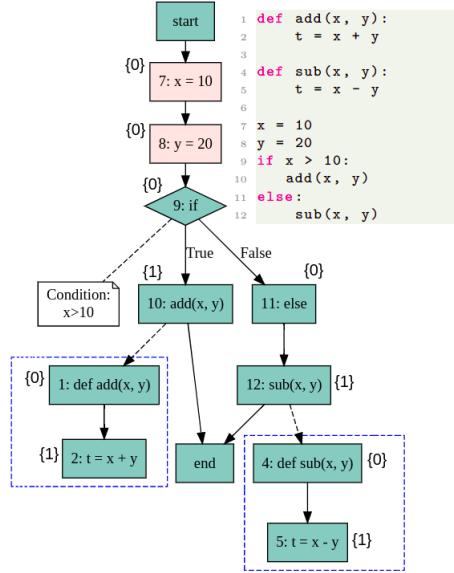


Figura 4.12: Diagrama de proveniência prospectiva (com endentação).

- `python3 prospective.py --trial 1 --var 1`: gera um diagrama de proveniência prospectiva com dados de proveniência retrospectiva referente ao conteúdo das variáveis (ver Figura 4.13).

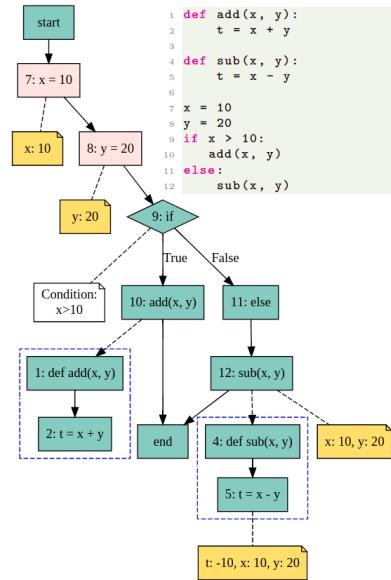


Figura 4.13: Diagrama de proveniência prospectiva e retrospectiva (somente conteúdo das variáveis).

- `python3 prospective.py --trial 1 --act 1`: gera um diagrama de proveniência prospectiva da *trial* 1 junto com dados de proveniência retrospectiva referente às ativações no código (ver Figura 4.14).

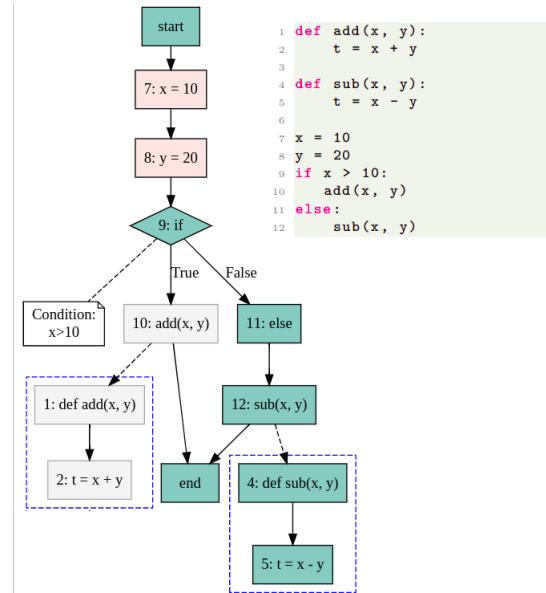


Figura 4.14: Diagrama de proveniência prospectiva e retrospectiva (somente ativações).

- `python3 prospective.py --trial 1 --show 1 7 8`: gera um diagrama de proveniência prospectiva do código compreendido entre a linha 7 e a linha 8 (ver Figura 4.15).

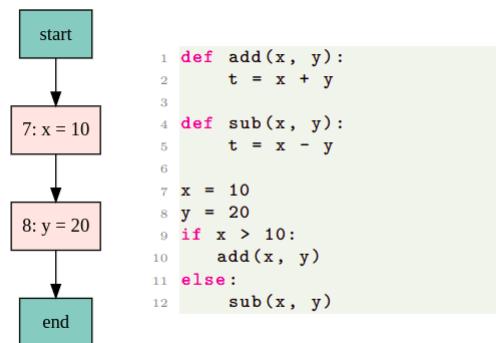


Figura 4.15: Diagrama de proveniência prospectiva (parte do código).

- `python3 prospective.py --trial 1 --show f sub`: gera um diagrama de proveniência prospectiva mostrando apenas a função `sub` (ver Figura 4.16).

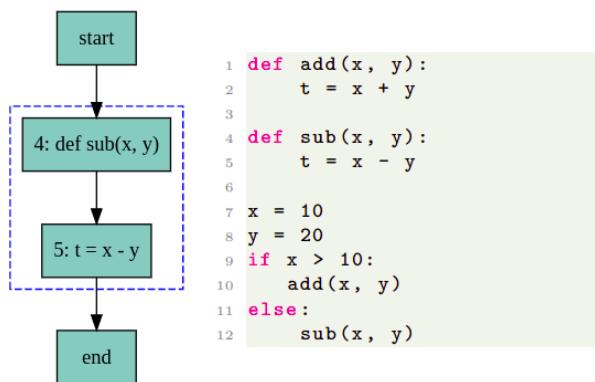


Figura 4.16: Diagrama de proveniência prospectiva (exibindo apenas uma função).

## 4.8 Considerações Finais

Este capítulo apresentou uma descrição detalhada da arquitetura do mecanismo de geração de grafos de proveniência prospectiva, o ProspectiveProv. O ProspectiveProv foi desenvolvido para atuar em conjunto com o noWorkflow, permitindo gerar grafos de proveniência prospectiva de *scripts* Python. Nesse sentido, destacamos algumas das principais características do ProspectiveProv:

- Permite gerar grafos de proveniência de *script* automaticamente, dependendo apenas dos dados de proveniência coletados e catalogados pelo noWorkflow;
- Representa os dados utilizando um modelo visual simplificado, utilizando fluxogramas de algoritmo ao invés de notações de proveniência para facilitar a compreensão da estrutura do *script* e minimizar a quantidade de nós produzidos no grafo do *script*;
- Permite combinar dados de proveniência prospectiva e retrospectiva em um mesmo grafo de proveniência.

O capítulo seguinte mostra detalhes do estudo experimental criado como forma de avaliar a abordagem ProspectiveProv, comparando-a a outras ferramentas similares que permitem gerar grafos de proveniência prospectiva de *scripts*. São apresentados os resultados obtidos com este estudo e uma descrição das possíveis ameaças à validade.

# Capítulo 5

## Estudo Experimental

### 5.1 Introdução

Neste estudo experimental, pretende-se caracterizar o apoio existente à compreensão da estrutura de *scripts* através de diagramas de proveniência gerados pelo ProspectiveProv. Para isso, o experimento foi realizado com participantes voluntários, que usaram o ProspectiveProv e outras ferramentas para realizar algumas tarefas, em duas fases distintas. Na Fase 1 do experimento, o ProspectiveProv é comparado a outras duas abordagens com o mesmo propósito: o Visustin [46], e o ProvenanceCurious [32, 31]. Tais abordagens foram escolhidas pois ambas permitem gerar diagramas da estrutura de *scripts* desenvolvidos em Python, e não necessitam de nenhuma anotação do usuário para funcionar. Além disso, tanto o Visustin quanto o ProvenanceCurious são ferramentas focadas em melhorar o entendimento da estrutura de códigos usando diagramas. Na Fase 2 do experimento, o ProspectiveProv é comparado ao uso isolado de *scripts* (sem o uso de diagramas).

No contexto desse estudo, são avaliadas a eficácia e a eficiência na análise e na compreensão de scripts usando a abordagem ProspectiveProv. A eficiência é medida calculando-se a duração de um conjunto de tarefas, enquanto a eficácia é medida usando-se o total de acertos dos participantes. Desse modo, esse estudo pretende responder as seguintes questões de pesquisa.

- **Questão de Pesquisa 1:** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficazes na análise e compreensão da estrutura de um *script* quando comparado a outras abordagens semelhantes?
- **Questão de Pesquisa 2:** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficazes na análise e compreensão da estrutura de um *script*

quando comparado a não usar esse tipo de ferramenta?

- **Questão de Pesquisa 3:** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficientes na análise e compreensão da estrutura de um *script* quando comparado a outras abordagens semelhantes?
- **Questão de Pesquisa 4:** Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficientes na análise e compreensão da estrutura de um *script* quando comparado a não usar esse tipo de ferramenta?

Este capítulo está organizado como segue. A Seção 5.2 apresenta detalhes sobre o design geral do experimento. A Seção 5.3 apresenta os resultados obtidos nesse estudo. A Seção 5.5 discute as possíveis ameaças à validade do experimento. Por fim, a Seção 5.6 apresenta as considerações finais.

## 5.2 Design do Experimento

Devido à pandemia de COVID-19, os experimentos precisaram passar por reformulações de forma a garantir a segurança dos participantes e ao mesmo tempo uma estrutura para mitigar possíveis ameaças a validade. Diante disso, o experimento foi conduzido online, por meio de formulários Web e uso de *plugins* para a medição do tempo de execução. Para permitir isso, os experimentos foram executados usando o Jotform<sup>1</sup>, uma plataforma online para desenvolvimento de formulários e que permite adicionar regras condicionais, *plugins* e integrações com outras aplicações.

Antes da versão definitiva do experimento, foram aplicados quatro testes piloto (em sessões diferentes) com alunos e pesquisadores da Universidade Federal Fluminense. Estes testes são importantes para ajustar o experimento e identificar problemas durante a sua execução, além de detectar pontos de falha. Para este estudo, até a versão final, quatro modificações foram realizadas, seguindo critérios de avaliação como forma de minimizar os impactos das ameaças à validade, que são descritas mais detalhadamente na Seção 5.5. Além disso, para assegurar que os tempos de execução não são outliers e controlar e mitigar as ameaças à validade, foram criados critérios quanto ao design do experimento online, conforme discutido a seguir.

- **Quanto aos acessos,** os participantes receberam um e-mail com convites para a

---

<sup>1</sup><https://jotform.com>

participação do experimento. Ao concordarem, o redirecionamento é automático e a seleção de grupos é realizado por um algoritmo de *round-robin*.

- **Quanto aos critérios de seleção**, apenas os participantes com alguma experiência em Python podem participar do experimento. Qualquer participante que não se encaixe neste critério é automaticamente descartado.
- **Quanto a comunicação entre participantes**, nenhum participante sabe de qual grupo ele participou. Além disso, os dados dos participantes são anonimizados, nenhuma informação pessoal do participante é utilizada ou armazenada.
- **Quanto a participantes duplicados**, o JotForms possui recursos para bloquear submissões duplicadas. Nesse caso, foram ativados os filtros para submissões únicas, que verifica os *cookies* e o endereço de IP para impedir que os participantes que já submeteram respostas das tarefas do experimento façam novas submissões.
- **Quanto ao questionário de consentimento**, os participantes recebem estes formulários antes de iniciar o estudo. Caso aceitem, eles são redirecionados automaticamente para o experimento.
- **Quanto as instruções**, existe uma caixa de texto com instruções detalhadas para cada tarefa como forma de chamar a atenção do participante para critérios importantes. Nesse momento, nenhum participante conhece as abordagens que são avaliadas nesse estudo.
- **Quanto as respostas obrigatórias**, alguns formulários desse estudo possuem campos obrigatórios, são eles, o formulário de consentimento e o questionário de caracterização.
- **Quanto as respostas em branco**, são permitidos respostas em branco para as tarefas.
- **Quanto as tarefas**, tanto os diagramas de avaliação quanto os *scripts* estão em formatos de imagem nos formulários. Utilizar imagens no lugar do texto tem o propósito de evitar que os participantes copiem e coleem as tarefas e façam buscas externas, o que pode afetar os resultados finais desse estudo.
- **Quanto aos botões de avançar e retroceder nos formulários**, foram bloqueados os botões de retroceder para evitar que os participantes alterem as respostas

após completá-las. No entanto, os participantes podem avançar para a próxima tarefa mesmo com questões em branco. Apenas informações obrigatórias (questionário de caracterização) não possibilitam o candidato a prosseguir sem preencher-las.

- **Quanto ao cronômetro**, foi inserido um *timer* isolado para cada tarefa (inserido como *plugin*). O participante não possui acesso ao *timer*, mas ele é iniciado ao abrir uma tarefa e parado ao seu término. Tais cronômetros são individuais, eles não interferem no funcionamento dos outros.
- **Quanto ao questionário de avaliação**, foram criados questionários de avaliação com perguntas sobre as tarefas. Tais questionários aparecem ao término de cada fase do experimento.
- **Quanto a submissão**, elas são feitas ao término do experimento e ficam armazenadas na plataforma Jotform. Nenhum participante possui acesso a elas.
- **Quanto a violação**, os participantes que violaram critérios para a realização das tarefas são eliminados e descartados, não sendo inseridos nas análises finais ou em qualquer outra discussão de resultados.

Os experimentos foram projetados com o propósito de não sofrerem interferências, isto é, os participantes devem realiza-lo continuamente, sem qualquer parada até a sua submissão. Essa restrição foi apresentada aos participantes antes de iniciar o experimento. Uma descrição do passo-a-passo de experimento é fornecida a seguir.

### Fase Inicial

1. O participante responde a um formulário de caracterização.
2. O participante é colocado em um grupo com base em sua experiência acadêmica e experiência em Python.
3. O participante recebe instruções do estudo na tela e o formulário de consentimento.
4. O participante recebe instruções básicas para realizar as tarefas.

### Fase 1

1. O participante recebe instruções para as tarefas da Fase 1.

2. O participante inicia a tarefa  $A_1$  - ele recebe uma imagem de um diagrama e a questão que deve ser respondida. O cronômetro oculto é iniciado. O participante fornece a resposta para a tarefa e clica em um botão para ir para a próxima tarefa, então o cronômetro é finalizado e não é mais possível voltar a esta tarefa.
3. A etapa acima se repete para as tarefas  $A_2$  e  $A_3$ .
4. Após terminar a tarefa  $A_3$ , o participante recebe um questionário de avaliação. O participante avalia as questões que acabou de responder.

## Fase 2

1. O participante recebe instruções para as tarefas da Fase 2.
2. O participante inicia a tarefa  $B_1$  - ela recebe um diagrama ou *script* e a pergunta que deve ser respondida. O cronômetro oculto é iniciado. Ela fornece a resposta para a tarefa e clica em um botão para ir para a próxima tarefa, então o cronômetro termina. Depois desse ponto, não é mais possível voltar a essa tarefa.
3. A etapa acima se repete para a tarefa  $B_2$ .
4. O participante recebe um questionário de avaliação para as tarefas da Fase 2.
5. O participante envia suas respostas e o experimento é concluído.

A Figura 5.1 mostra o fluxograma de como os participantes realizam o experimento do inicio até o momento de submissão das respostas. Vale lembrar que inicialmente eles recebem as tarefas relacionadas à Fase 1, e após a conclusão, logo em seguida, iniciam as tarefas da Fase 2. A Figura 5.2 mostra o fluxograma de exemplo pra uma fase qualquer do experimento, visto que elas seguem o mesmo ciclo. Além disso, como os cronômetros são ocultos, nenhum participante sabe o tempo que ele levou para terminar cada tarefa.

### 5.2.1 Fase 1: Comparação das abordagens do estudo

Nesta fase do experimento são avaliadas a eficácia e eficiência do ProspectiveProv com relação as demais abordagens citadas anteriormente. A Fase 1 é constituída por três tarefas: *tarefa  $A_1$* , *tarefa  $A_2$* , e *tarefa  $A_3$* .

Nessa fase, o propósito é avaliar o tempo que os participantes levam para compreender um diagrama e re-escrever o código correspondente a ele, além de avaliar se a resposta

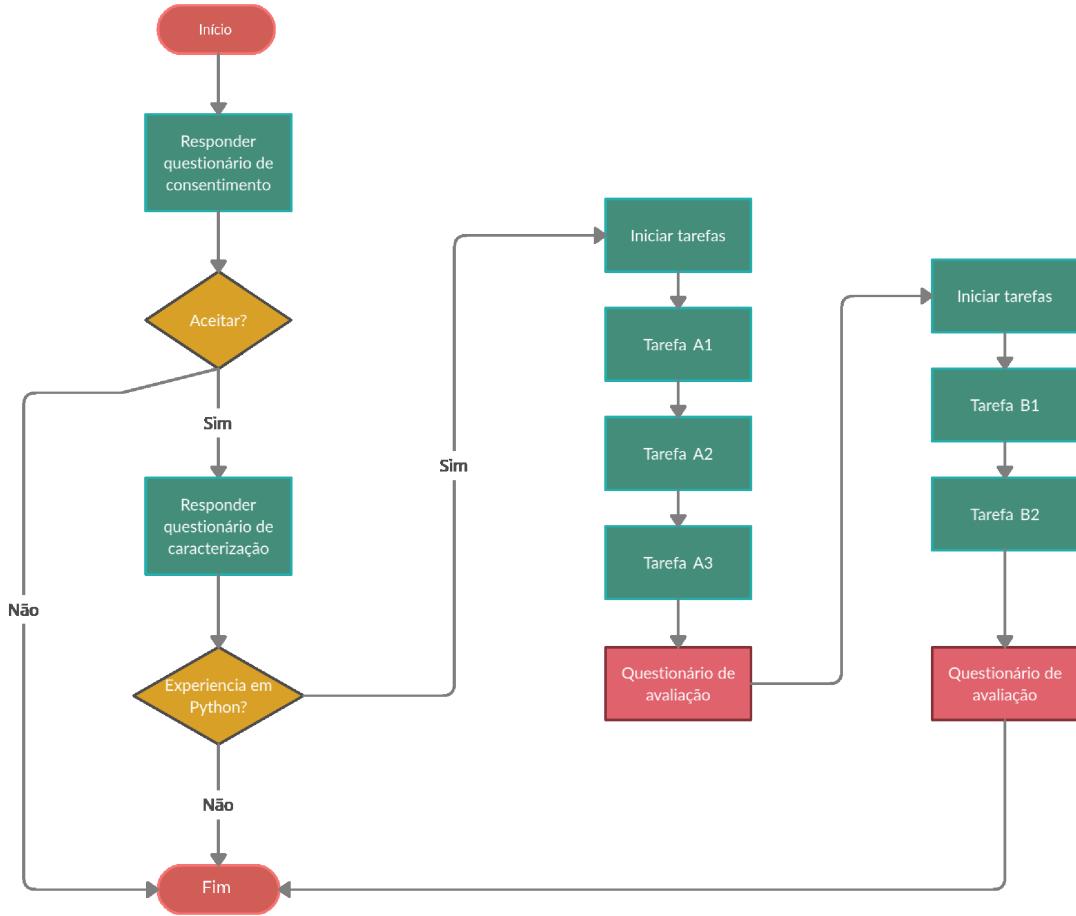


Figura 5.1: Fluxograma relacionado aos processos envolvidos no experimento. Tarefas  $A_1$ ,  $A_2$  e  $A_3$  são tarefas da Fase 1 do experimento, enquanto as tarefas  $B_1$  e  $B_2$  pertencem à Fase 2.

fornecida está correta. Essa avaliação ajuda a analisar se os diagramas são intuitivos o suficiente para o participante conseguir replicá-los em termos de código. Nesse experimento comparativo, os participantes recebem tarefas com níveis de dificuldade diferentes. A Tabela 5.1 descreve quais são os conceitos envolvidos em cada tarefa.

Para distribuir as tarefas da Fase 1, foi utilizado o esquema de quadrados latinos [45] para as três abordagens (ProvenanceCurious, ProspectiveProv e Visustin), conforme mostra a Tabela 5.2. Para produzir estes grupos, foi utilizado a resposta do questionário de caracterização combinado a uma heurística para a distribuição dos participantes, buscando obter grupos mais homogêneos nas duas fases do experimento. Tal heurística utiliza o algoritmo de *round-robin* e prioriza experimentos com participantes ausentes com determinados perfis de caracterização para prevenir que os grupos se tornem desequilibrados ou muito heterogêneos. Assim, para a Fase 1 desse estudo, dividimos os participantes em três grupos ( $X_1$ ,  $X_2$  e  $X_3$ ). Além disso, seguindo o desenho do quadrado latino, cada

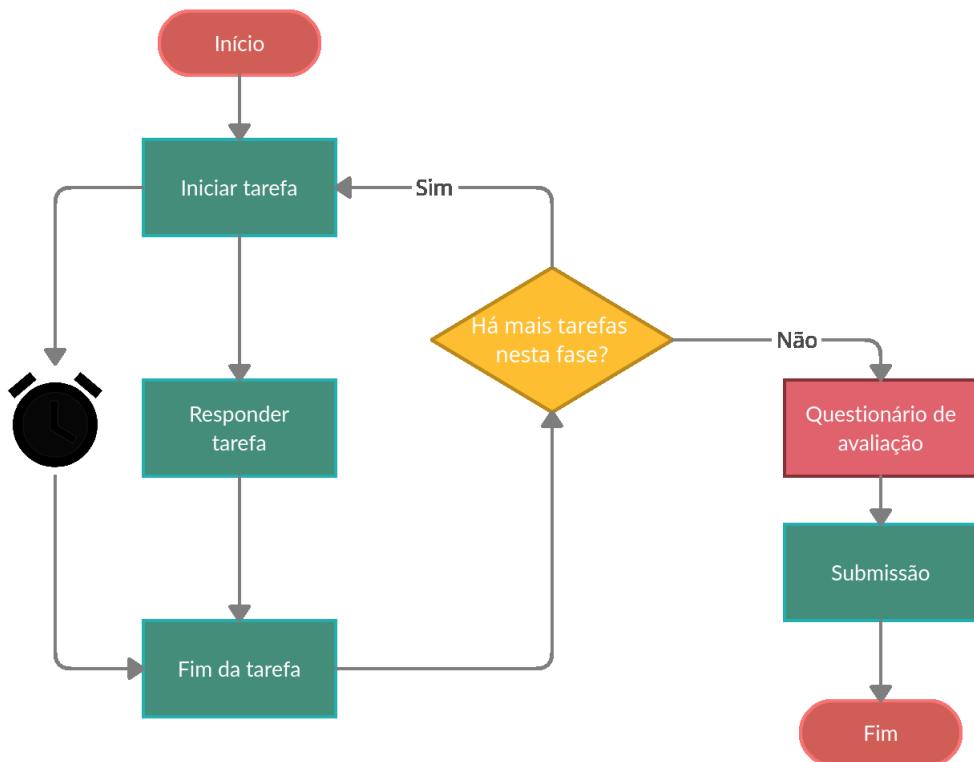


Figura 5.2: Fluxograma relacionado aos processos envolvidos numa fase.

Tabela 5.1: Conhecimento em Python necessário para conclusão das tarefas da Fase 1.

Fase 1		
Tarefa A <sub>1</sub>	Tarefa A <sub>2</sub>	Tarefa A <sub>3</sub>
<i>if-else</i>	<i>loops</i>	funções e chamadas

Tabela 5.2: Esquema de quadrados latinos (Fase 1).

Grupo	Tarefa A <sub>1</sub>	Tarefa A <sub>2</sub>	Tarefa A <sub>3</sub>
X <sub>1</sub>	ProspectiveProv	ProvenanceCurious	Visustin
X <sub>2</sub>	Visustin	ProspectiveProv	ProvenanceCurious
X <sub>3</sub>	ProvenanceCurious	Visustin	ProspectiveProv

grupo recebeu uma ferramenta diferente para cada uma das tarefas. Por exemplo, o grupo X<sub>1</sub> usou ProspectiveProv para realizar tarefa A<sub>1</sub>, ProvenanceCurious para a tarefa A<sub>2</sub> e Visustin a tarefa A<sub>3</sub>. Para os outros grupos, a configuração mudou de acordo com o desenho do quadrado latino. Todos os diagramas e tarefas desse experimento podem ser consultados nos Apêndices A e C.

### 5.2.2 Fase 2: Comparação do uso de scripts e diagramas

Na Fase 2 do estudo são avaliados a eficácia e eficiência dos participantes que usam apenas os scripts para analisar um experimento e os que usam apenas os diagramas gerados pelo ProspectiveProv. Nesta fase, existem duas tarefas, uma utilizando os *scripts* (*tarefa B<sub>1</sub>*) e outra utilizando os diagramas (*tarefa B<sub>2</sub>*).

O propósito dessa fase é mensurar o tempo que os participantes levam para analisar e entender o que acontece em um dado ponto do *script* ou do diagrama, além de medir se as respostas estão corretas. Esta avaliação auxilia a compreender se os participantes acertam mais usando somente a estrutura de um código, ou por meio de um diagrama que representa um outro código similar.

Para este experimento, foi criada uma situação onde os participantes devem responder o que uma determinada parte do diagrama ou do código retorna. Para isso, os participantes recebem um valor de entrada e este valor deve ser analisado com base no código ou diagrama fornecido. Dessa forma, pretende-se entender qual é a melhor abordagem para que os participantes identifiquem e compreendam as operações que o código ou diagrama representam. Essa fase do experimento está descrita nos Apêndices B e D.

Para a tarefa de diagramas, os participantes também recebem um valor de entrada. O objetivo é que os participantes respondam o que uma determinada parte do diagrama retorna. Para as tarefas da Fase 2, também foi utilizado o esquema de quadrados latinos, agora usando duas abordagens (*script* e diagramas do ProspectiveProv), conforme é exibido na Tabela 5.3. Dessa maneira, na Fase 2, os participantes foram divididos em dois grupos ( $Y_1$  e  $Y_2$ ). Além disso, a Tabela 5.4 descreve o nível de conhecimento necessário em Python para a resolução das tarefas da Fase 2.

Tabela 5.3: Esquema de quadrados latinos (Fase 2).

Grupo	Tarefa <i>B<sub>1</sub></i>	Tarefa <i>B<sub>2</sub></i>
$Y_1$	<i>Script</i> Python	Diagrama
$Y_2$	Diagrama	<i>Script</i> Python

### 5.2.3 Perfil dos participantes

Esta seção apresenta a avaliação detalhada do perfil dos participantes que aceitaram fornecer os dados do questionário de caracterização para avaliação nesse estudo. Tais

Tabela 5.4: Conhecimento em Python necessário para conclusão das tarefas da Fase 2.

<b>Fase 2</b>	
<b>Tarefa <math>B_1</math></b>	<b>Tarefa <math>B_2</math></b>
<i>if-else, loops, funções e chamadas</i>	<i>if-else, loops, funções e chamadas</i>

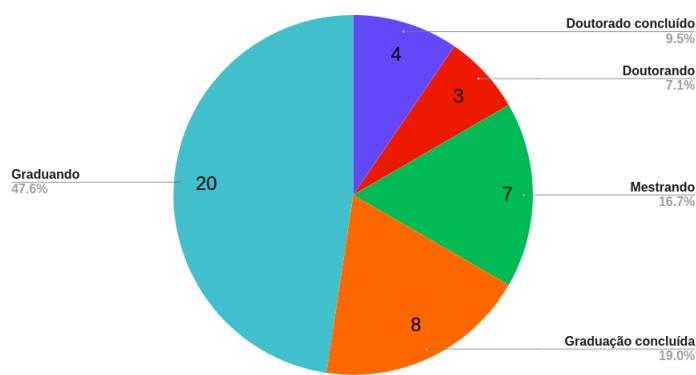


Figura 5.3: Participantes divididos por experiência acadêmica.

dados são importantes para compreender a relação entre os participantes e as tarefas respondidas por eles.

Para responder as questões de pesquisa apresentadas anteriormente, foram convidados alunos dos diversos cursos de graduação e pós-graduação oferecidos pelo Departamento de Ciência da Computação da Universidade Federal Fluminense. Além desses, foram convidados candidatos externos de outras instituições e desenvolvedores autônomos com conhecimento em Python. De todos os convites, no total, 42 participantes aceitaram participar deste estudo, dos quais 20 são alunos de graduação, 8 são graduados, 7 são alunos de mestrado, 3 são alunos de doutorado e 4 têm doutorado, conforme ilustrado na Figura 5.3. Como o número de graduandos se sobressai aos demais, muitos deles podem não ter experiência com diagramas de proveniência ou de fluxogramas.

A Figura 5.4 apresenta a experiência dos participantes separadas por área do conhecimento. A maioria dos participantes é composta por pessoas da área de computação. Detalhadamente, 88.1% dos participantes são de áreas relacionadas a computação, tais como redes de computadores, ciências da computação, sistemas de informação e engenharia de software. Outras áreas do conhecimento correspondem a 11.9% dos participantes, sendo estes de áreas como Ciências Econômicas (4.8%), Engenharia de Produção (2.4%), Linguística (2.4%) e Engenharia Biomédica (2.4%).

A Figura 5.5 apresenta a experiência dos candidatos com relação à linguagem Python.

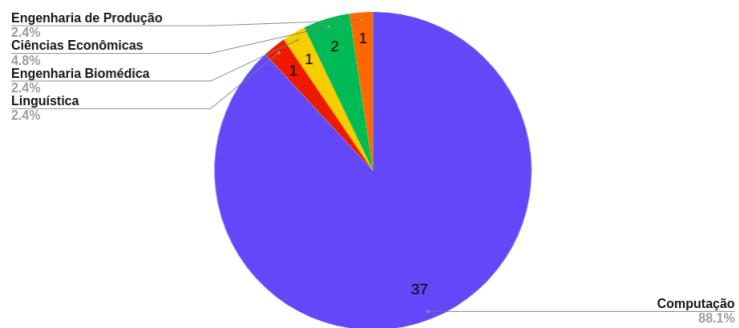


Figura 5.4: Participantes divididos por área do conhecimento.

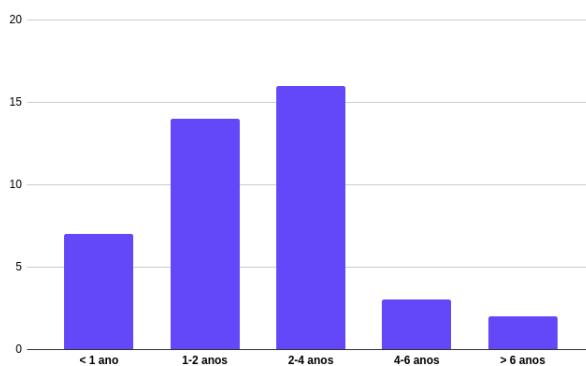


Figura 5.5: Participantes divididos por experiência em Python.

Nota-se que grande parte dos participantes tem entre 2 e 4 anos de experiência em Python, enquanto os participantes com mais de 6 anos de experiência são a minoria. Por fim, a Figura 5.6 descreve os anos de experiência dos participantes em desenvolvimento de software. A experiência é separada em setores, como pessoal, acadêmico, open-source e indústria. A experiência pessoal refere-se a anos com desenvolvimento de software de forma informal (desenvolvedores autônomos, projetos pessoais), acadêmico refere-se a anos de experiência relacionados a área acadêmica (como projetos de pesquisa por exemplo), open-source está relacionado a desenvolvimento de software livre, e indústria refere-se a experiência profissional (desenvolvimento de software para empresas).

## 5.3 Resultados e Discussões

Apresentamos nessa seção a avaliação de resultados desse estudo. A avaliação foi realizada a partir do ambiente de *software* livre R<sup>2</sup>, através do RStudio Cloud<sup>3</sup>, um serviço gratuito oferecido na nuvem que permite que usuários executem projetos R na nuvem. Para este

<sup>2</sup><https://www.r-project.org>

<sup>3</sup><https://rstudio.cloud>

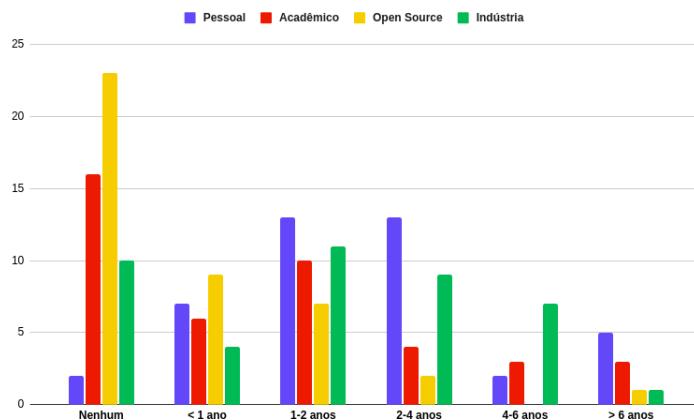


Figura 5.6: Participantes divididos por experiência em software.

estudo, as variáveis utilizadas são o número de acertos totais e a duração.

### 5.3.1 Eficácia

A eficácia foi analisada com base no número de respostas corretas de cada tarefa. Em termos de eficácia, para cada resposta respondida corretamente, o participante recebe um ponto ( $t=1$ ) e zero pontos para respostas incorretas ( $t=0$ ). Para as tarefas da Fase 1 ( $A_1$ ,  $A_2$  e  $A_3$ ), é considerado como resposta correta apenas se o participante replicou corretamente o código-fonte correspondente ao diagrama avaliado naquela tarefa. Da mesma forma, para as tarefas da Fase 2 ( $B_1$  e  $B_2$ ), consideramos uma resposta como correta quando o participante corretamente identificou os valores retornados pelas funções usando o diagrama gerado pelo ProspectiveProv ou código Python. Como as tarefas  $B_1$  e  $B_2$  da Fase 2 são as mesmas (uma tarefa analisando um código e outra analisando um diagrama, apenas aplicadas em ordens diferentes), na análise dos resultados, combinamos os resultados. Dessa forma, na nossa análise, a tarefa  $B$  corresponde à união dos resultados das tarefas  $B_1$  e  $B_2$ . Essa união não interfere nos resultados – ela apenas facilita as análises estatísticas do presente estudo.

A Figura 5.7 mostra os gráficos de barras do número total de respostas corretas de cada tarefa neste estudo experimental. A cor verde indica as respostas corretas e a vermelha indica o número de respostas incorretas. Ao analisar a Figura 5.7, é possível observar que nas tarefas comparativas entre as abordagens (Fase 1), os participantes que utilizaram o Visustin obtiveram maior número de acertos, em segundo o ProspectiveProv e em terceiro o ProvenanceCurious. Nota-se que o ProvenanceCurious obteve uma taxa de erros muito maior se comparado às outras abordagens.

Quanto a análise comparativa entre *scripts* e os diagramas do ProspectiveProv (Fase 2), os participantes que utilizaram apenas o *script* obtiveram pontuações ligeiramente maiores. É importante notar que a diferença entre o número total de respostas corretas usando os diagramas ProspectiveProv e o *script* é de apenas 3 pontos (com um total de 42 participantes). Além disso, 60% dos participantes de áreas não computacionais sentiram-se mais confortáveis com o diagrama ProspectiveProv do que com o *script*, de acordo com as respostas que eles forneceram nos questionários de avaliação após a conclusão das tarefas dessa Fase.

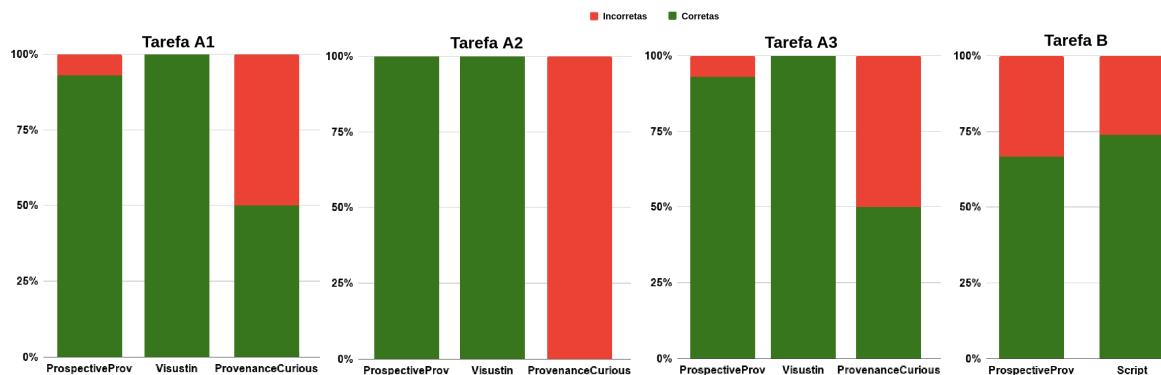


Figura 5.7: Taxa de respostas corretas.

Para medir a significância estatística do número total de respostas corretas, aplicamos o método de Fisher para comparar as abordagens 2x2 [16], uma vez que a variável nesse caso, é categórica (acerto ou erro). A Tabela 5.5 mostra os resultados do teste de Fisher. Nesse caso, há uma diferença estatisticamente significativa quando comparamos ProspectiveProv e ProvenanceCurious nas tarefas  $A_1$ ,  $A_2$  e  $A_3$ , comportamento que também ocorre entre Visustin e ProvenanceCurious. No entanto, a diferença entre os resultados do ProspectiveProv e Visustin não é estatisticamente significativa. Assim, é possível concluir que ambos são igualmente eficazes neste caso. O mesmo ocorre para os resultados dos diagramas do ProspectiveProv comparado aos *scripts*.

Por meio das análises estatísticas realizadas sobre os dados de eficácia apresentadas nessa seção, as questões de pesquisa relacionadas à eficácia podem ser respondidas da seguinte forma.

**Questão de Pesquisa 1:** *Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficazes na análise e compreensão da estrutura de um script quando comparados a outras abordagens semelhantes?*

**Resposta:** *Não, no entanto, os participantes que realizaram as tarefas usando ProspectiveProv sentiram-se mais confortáveis com o diagrama ProspectiveProv do que com o script.*

Tabela 5.5:  $P$ -value para o teste de Fisher para o número total de respostas corretas. A hipótese nula é que não há diferença entre as abordagens comparadas. Valores azuis indicam  $p$ -value  $\leq \alpha$ , com  $\alpha = 0.05$ .

Comparação 2x2	Tarefa	$p$ -value
ProspectiveProv e Visustin	A1	$\approx 1.0000$
	A2	$\approx 1.0000$
	A3	$\approx 1.0000$
ProspectiveProv e ProvenanceCurious	A1	0.0329
	A2	<0.0001
	A3	0.0329
Visustin e ProvenanceCurious	A1	0.0058
	A2	<0.0001
	A3	0.0058
Diagrama e <i>Script</i>	B	0.6337

*utiveProv obtiveram o segundo maior número de respostas corretas, exceto para a tarefa A<sub>2</sub>, onde Visustin e ProspectiveProv têm taxas de sucesso semelhantes. Além disso, não há diferença estatisticamente significativa entre os resultados de Visustin e ProspectiveProv para todas as tarefas, e ambos foram mais eficazes do que ProvenanceCurious (como mostrado na Tabela 5.5). Visustin e ProspectiveProv usam diagramas baseados em fluxogramas, enquanto ProvenanceCurious usa uma notação própria de proveniência. Nesse aspecto, os participantes tiveram pior desempenho usando notações de proveniência. Comparado ao Visustin, ProspectiveProv tem a vantagem de usar dados de proveniência de execução para construir o diagrama, e assim garantir que o diagrama reflita um código correto, enquanto Visustin não é capaz de garantir isso. Além disso, ProspectiveProv também é capaz de incluir dados de proveniência retrospectiva diretamente no diagrama.*

**Questão de Pesquisa 2:** *Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficazes na análise e compreensão da estrutura de um script quando comparados a não usar esse tipo de ferramenta (apenas usando o código)?*

**Resposta:** *Não, os participantes que realizaram as tarefas usando apenas o código do script obtiveram pontuações mais altas do que os participantes que usaram os diagramas de código. No entanto, essa diferença não é estatisticamente significativa (conforme mostrado na Tabela 5.5). Portanto, pode-se dizer que ambas as abordagens são igualmente eficazes.*

**Comentário:** *Vale ressaltar que, com base no questionário final de avaliação, 60% dos participantes de áreas diferentes da Ciência da Computação expressaram maior preferê-*

*cia por diagramas ao invés de código de script.*

### 5.3.2 Eficiência

Nesta seção apresentamos os resultados e discussões quanto a eficiência. Para medir a eficiência, foi medido o tempo total (duração) das tarefas para as quais os participantes produziram respostas corretas, uma vez que não faz sentido produzir respostas rápidas, porém incorretas. Vale destacar que na comparação entre as abordagens (Fase 1 do estudo), o ProvenanceCurious será altamente afetado, com vários resultados que precisaram ser descartados, já que ele obteve uma alta taxa de respostas incorretas nessa fase (como mostrado anteriormente na Figura 5.7). Assim como para a análise de eficácia, os dados de duração das tarefas  $B_1$  e  $B_2$  também foram analisados em conjunto, correspondendo à tarefa  $B$  nas figuras dessa seção.

A Figura 5.8 mostra os boxplots para a variável de duração e resume os resultados obtidos para as tarefas da Fase 1, já descartando os tempos das respostas incorretas. Para enriquecer a avaliação de resultados, a Figura 5.8 também mostra as métricas de média e desvio padrão para a variável de duração em segundos (considerando apenas as respostas corretas). A Figura 5.9 mostra os resultados obtidos e as métricas de média e desvio padrão para a variável de duração em segundos das tarefas da Fase 2.

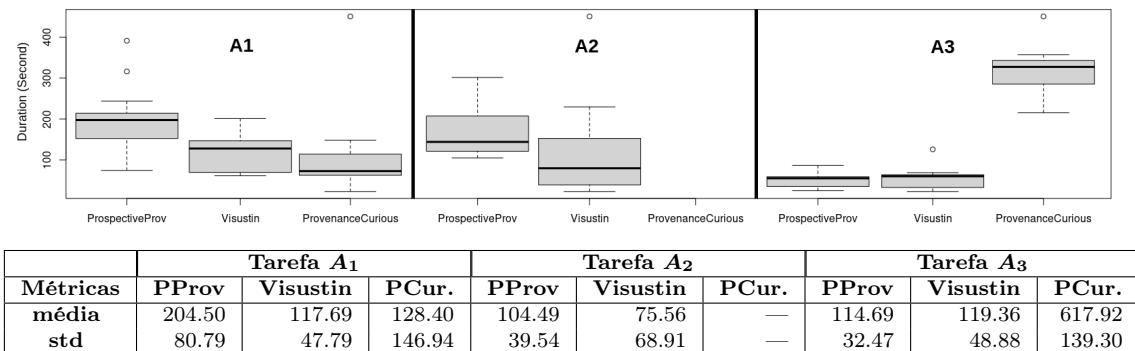


Figura 5.8: Análise da variável duração (Fase 1).

Para testar se há diferença estatisticamente significativa na eficiência das diferentes abordagens, usamos o Teste de Mann-Whitney para comparar as abordagens 2x2, uma vez que desejamos testar a diferença da distribuição de uma variável contínua (duração) entre duas populações. A Tabela 5.6 mostra os resultados. Para as tarefas da Fase 1, isto é,  $A_1$ ,  $A_2$  e  $A_3$ , existe uma diferença significativa entre ProvenanceCurious e as outras abordagens na maioria das tarefas, o que leva à conclusão de que ele é menos eficiente do

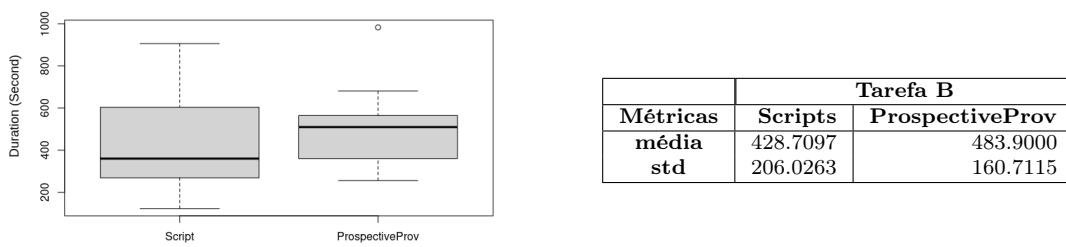


Figura 5.9: Análise da variável duração (Fase 2).

Tabela 5.6:  $P$ -value para o teste de Mann-Whitney para a variável duração. A hipótese nula é que não há diferença entre as abordagens comparadas. Os valores azuis indicam  $p$ -value  $\leq \alpha$ , com  $\alpha = 0,05$ .

Comparação 2x2	Tarefas	$p$ -value
ProspectiveProv e Visustin	A1	0.00244
	A2	0.06010
	A3	0.78716
ProspectiveProv e ProvenanceCurious	A1	0.02144
	A2	-
	A3	0.00036
Visustin e ProvenanceCurious	A1	0.43540
	A2	-
	A3	0.00030
Diagrama e <i>Script</i>	B	0.13622

que o ProspectiveProv e o Visustin. Para a comparação entre ProspectiveProv e Visustin, há uma diferença significativa apenas na Tarefa  $A_1$ , onde os participantes que utilizaram os diagramas do Visustin tiveram um desempenho melhor do que aqueles que usaram o ProspectiveProv. No caso da Fase 2, que compara os diagramas gerados pelo ProspectiveProv e os *scripts*, existe uma pequena diferença no tempo médio dos participantes que utilizaram apenas os códigos, porém esta diferença não é estatisticamente significativa.

Com relação na análise e resultados obtidos quanto a eficiência para as duas fases desse estudo, as questões de pesquisa podem ser respondidas da seguinte forma.

**Questão de Pesquisa 3:** *Os diagramas ProspectiveProv permitem que os usuários sejam mais eficientes na análise e compreensão da estrutura de um script quando comparados a outras abordagens semelhantes?*

**Resposta:** *Não, os participantes que usaram Visustin tiveram melhor desempenho que o ProspectiveProv na tarefa  $A_1$  em termos de eficiência. No entanto, para as tarefas restantes, o ProspectiveProv é tão eficiente quanto Visustin. É importante notar que a*

tarefa  $A_1$  é a mais simples de todas, enquanto as tarefas  $A_2$  e  $A_3$  são as mais complexas (não apresentando significativa diferença entre ambas as abordagens). A maior diferença comparativa ocorre entre o ProvenanceCurious e o ProspectiveProv, onde o ProspectiveProv se mostra mais eficiente. Além disso, é importante lembrar que os participantes que realizaram a tarefa  $A_2$  usando o ProvenanceCurious erraram as respostas, e portanto todas as amostras para essa tarefa foram descartadas.

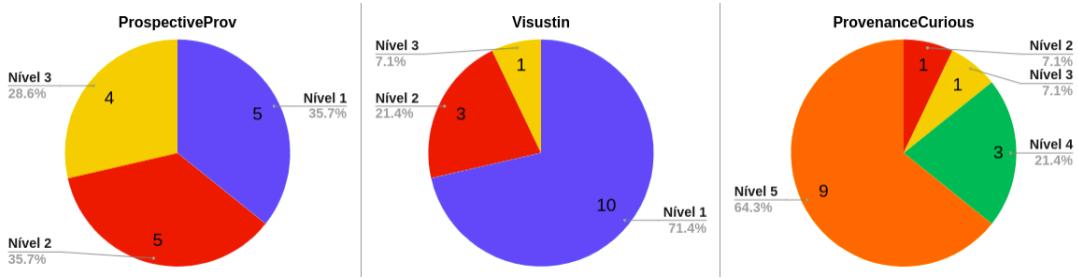
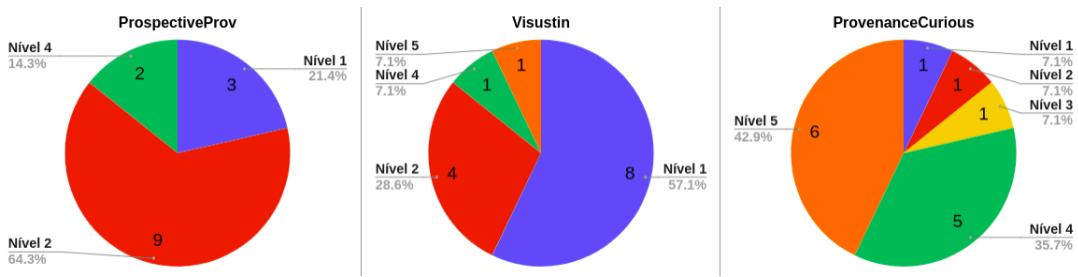
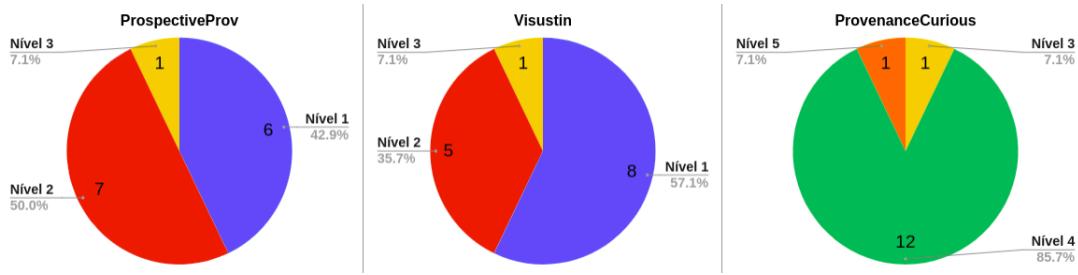
**Questão de Pesquisa 4:** *Os diagramas do ProspectiveProv permitem que os usuários sejam mais eficientes na análise e compreensão da estrutura de um script quando comparados a não usar esse tipo de ferramenta (apenas usando o código)?*

**Resposta:** *Não, a análise concluiu que não há diferença significativa entre os resultados. Porém, de acordo com o questionário final, os participantes de outras áreas do conhecimento (exceto Ciência da Computação e áreas afins) se sentiram mais à vontade em responder às questões envolvendo os diagramas.*

### 5.3.3 Questionário de avaliação dos participantes

Nesta seção são discutidos os resultados do questionário de avaliação dos participantes. Para compreender como os participantes se sentem ao responder as questões de estudo, elaboramos um questionário ao final de cada uma das fases. De maneira geral, foram analisadas cada resposta dos participantes quanto ao grau de dificuldade na execução das tarefas. O grau de dificuldade é medido em um intervalo de 1 à 5, onde 1 é o mais fácil e 5 o mais difícil.

A Figura 5.10 mostra a análise das respostas dos participantes quanto ao grau de dificuldade da tarefa  $A_1$  separado pelas abordagens avaliadas nesse estudo. Os participantes classificaram a tarefa com o Visustin a mais fácil, enquanto a tarefa com o ProvenanceCurious foi considerada a mais difícil entre as abordagens. No caso do ProspectiveProv, os participantes distribuíram igualmente a votação entre o nível 1 e o nível 2 de dificuldade. A Figura 5.11 mostra a mesma análise para a tarefa  $A_2$ . Para tal tarefa, dos participantes que utilizaram o Visustin, 57.1% classificaram os diagramas dele como extremamente fácil (nível 1). O ProspectiveProv foi considerado como fácil (nível 2) e o ProvenanceCurious obteve o nível de dificuldade extremamente difícil por 42.9% dos participantes. A Figura 5.12 mostra essa avaliação para a tarefa  $A_3$ . Nesse caso, o diagrama do Visustin também foi considerado o mais fácil (nível 1) pela maior parte dos participantes. O ProspectiveProv recebeu o nível de dificuldade 2 de 50% dos participantes. Já o ProvenanceCurious foi classificado como nível 4 por 85.7% dos participantes.

Figura 5.10: Grau de dificuldade na execução da tarefa  $A_1$ .Figura 5.11: Grau de dificuldade na execução da tarefa  $A_2$ .Figura 5.12: Grau de dificuldade na execução da tarefa  $A_3$ .

Diante dos resultados apresentados, a Tabela 5.7 resume como cada abordagem foi classificada levando em consideração cada uma das tarefas da Fase 1. Nessa avaliação, a classificação final foi definida com base na maior quantidade de votos que cada abordagem recebeu quanto ao grau de dificuldade das tarefas. Apesar do ProspectiveProv ter recebido em média 1 nível de dificuldade a mais do que o Visustin, o nível 2 ainda é considerado fácil.

Para complementar a análise descritiva relacionada ao questionário de avaliação dos participantes quanto ao nível de dificuldade das questões, foram selecionadas as respostas dos participantes da tarefa  $B$  (comparativa entre *scripts* e diagramas). Dessa forma, para a tarefa  $B$ , existe uma questão sobre qual abordagem os participantes preferem, sendo as opções possíveis, *script* ou diagramas. A Figura 5.13 mostra o nível de dificuldade considerado pelos participantes ao avaliar a tarefa  $B$ . É possível notar 38.1% dos participantes classificaram a tarefa utilizando *scripts* com nível de dificuldade 3, enquanto 45.2% dos

Tabela 5.7: Nível de dificuldade com base nas abordagens avaliadas (Fase 1).

Abordagem	Tarefa A1	Tarefa A2	Tarefa A3
ProspectiveProv	Entre o nível 1 e 2	Nível 2	Nível 2
Visustin	Nível 1	Nível 1	Nível 1
ProvenanceCurious	Nível 5	Nível 5	Nível 4

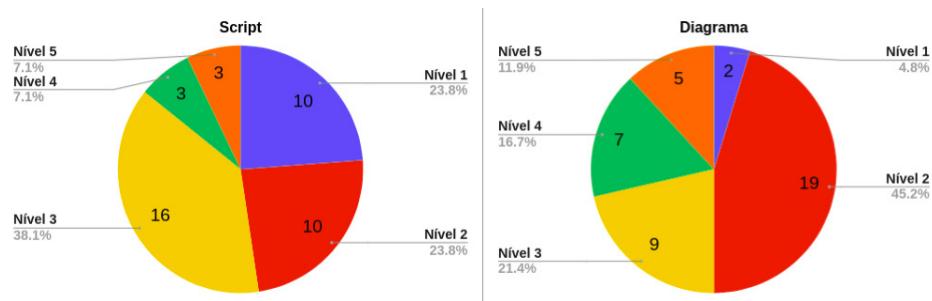


Figura 5.13: Grau de dificuldade na execução da tarefa B.

participantes que utilizaram apenas o diagrama do ProspectiveProv classificaram a tarefa com nível 2 de dificuldade. Diante disso, a Tabela 5.8 resume a classificação das tarefas da Fase 2.

Por fim, a Figura 5.14 ilustra a porcentagem de indivíduos que preferem *script* em comparação aos diagramas das tarefas realizadas nesse estudo. Nessa análise, observa-se que para os candidatos vinculados à área de computação, 100% deles preferem utilizar o *script*, enquanto para os participantes de outras áreas do conhecimento esse valor é de 40%. Dessa maneira, nota-se que participantes com menor conhecimento na área da computação são mais resilientes no uso de diagramas para compreensão da estrutura de códigos.

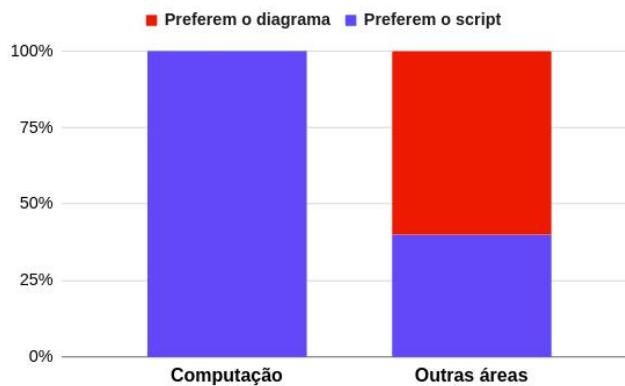
## 5.4 Avaliação de corretude

Nesta seção são discutidos os resultados quanto a avaliação de corretude de diagramas de *scripts* Python utilizando a abordagem Visustin. Embora o Visustin tenha tido bons resultados quando comparado às abordagens avaliadas no experimento anterior, tal ferramenta não valida ou executa os códigos antes de gerar os diagramas. Por este motivo, este experimento visa analisar como o Visustin se comporta com *scripts* Python que contenham erros de sintaxe.

Tanto o ProspectiveProv quanto o ProvenanceCurious somente geram os diagramas

Tabela 5.8: Nível de dificuldade com base nas abordagens avaliadas (Fase 2).

Abordagem	Tarefa B
Utilizando <i>scripts</i>	Nível 3
Utilizando diagramas	Nível 2

Figura 5.14: Preferência por abordagens (comparação entre *scripts* e diagramas).

de proveniência a partir de *scripts* Python sintaticamente corretos. Dessa forma, o ProspectiveProv coleta apenas dados de proveniência após a execução em sucedida de um *script* no noWorkflow. Já o ProvenanceCurious utiliza uma AST (*Abstract Syntax Trees*), e por isso, somente códigos sintaticamente corretos permitem gerar os diagramas de proveniência. Dessa forma, dentre tais abordagens, o Visustin é o único que permite gerar diagramas sem validação do código.

Para demonstrar e avaliar isso, foram realizados testes utilizando *scripts* Python com diferentes níveis de erros sintáticos em sua estrutura. A Figura 5.15 mostra o diagrama gerado pelo Visustin e seu respectivo código, onde o marcador em vermelho indica a parte do código com erro e o marcador em verde indica os blocos que estão corretos. Nesse cenário, no *script*, a linha 2 possui uma sintaxe inválida, enquanto as linhas 4, 5, 6 e 7 possuem erros referentes ao espaçamento.

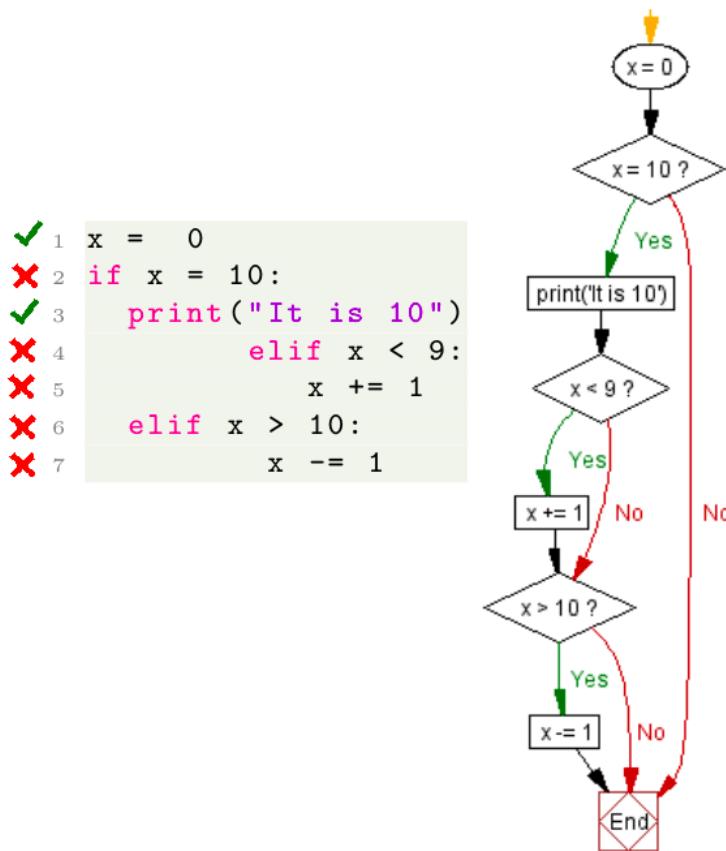


Figura 5.15: Teste de corretude utilizando laços condicionais.

Em outro cenário, foram testados códigos contendo declarações de *loops* com componentes não reconhecidos na linguagem Python. A Figura 5.16 mostra um diagrama e seu respectivo código Python utilizado para este teste. Nesse caso, apenas as linhas 2 e 5 possuem erros de sintaxe. Além disso, é possível notar que embora a estrutura do diagrama gerado esteja correta, existem erros visíveis com relação a declaração e condições de parada nos *loops* presentes no código.

Para o terceiro teste de validação, utilizamos um código contendo duas declarações de funções com retorno. A Figura 5.17 mostra o diagrama gerado pelo Visustin correspondente a este teste. Nesse exemplo, os erros de sintaxe estão na linha 1 e na linha 4. Na linha 1 existe um componente declarado de forma incorreta, enquanto na linha 4 existe um numeral definido incorretamente em meio ao bloco de retorno.

Diante dos testes apresentados, no Visustin, os usuários precisam garantir a corretude do seu próprio *script* Python antes de gerar os diagramas. Embora o Visustin consiga interpretar os principais componentes da estrutura do Python, os erros de sintaxe são refletidos diretamente nos diagramas gerados por ele. Nesse contexto, para erros quanto

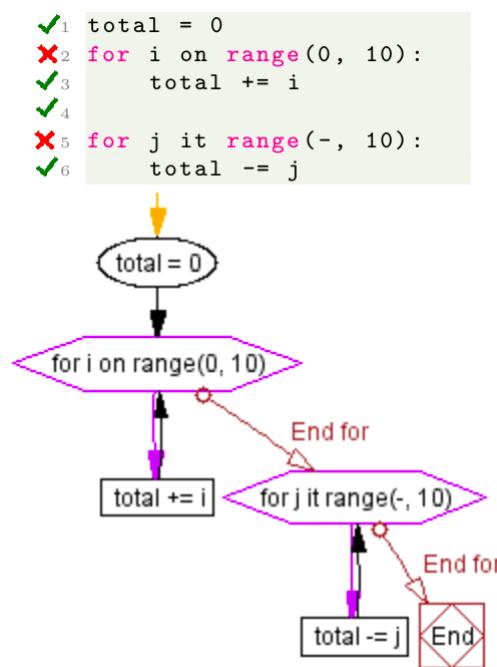


Figura 5.16: Teste de corretude utilizando *loops*.

a sintaxe, o Visustin os reflete nos textos dos nós do diagrama, enquanto os erros de espaçamento são ignorados completamente pela ferramenta.

## 5.5 Ameaças a validade

Durante o planejamento desta avaliação experimental, procuramos evitar ameaças que pudessem impactar ou limitar a validade de nossos resultados. Nesta seção, descrevemos as ameaças que identificamos nesta avaliação experimental.

Devido à pandemia de COVID-19, nosso estudo ocorreu por meio de formulários online por um período de um mês. O experimento ficou disponível durante este período e os participantes puderam acessá-lo a qualquer momento, dependendo de sua disponibilidade. No contexto de um estudo online, com tarefas construídas como formulários, não podemos garantir que estes participantes concluíram o estudo em um ambiente silencioso e sem interrupções durante o processo.

Com relação as tarefas, fornecemos aos participantes diferentes tarefas (em diversos níveis de dificuldade) para minimizar os efeitos do aprendizado em cada etapa deste estudo. O efeito de aprendizado pode ocorrer quando o participante realiza tarefas muito similares, o que pode resultar em um efeito memória, podendo afetar os resultados. Além disso, não informamos quais abordagens seriam avaliadas para evitar qualquer viés nas

```

✗ 1 def int add(x, y):
✓ 2     return x + y
✓ 3 def sub(x, y):
✗ 4     return 1x - y

```

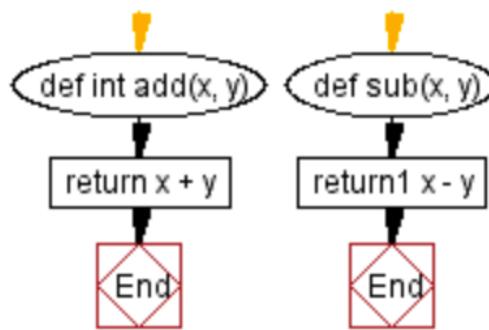


Figura 5.17: Teste de corretude utilizando funções.

respostas dos candidatos. Mesmo utilizando tais tratamentos, não podemos garantir a eliminação total desses efeitos e ameaças.

É importante mencionar as ameaças relacionadas a caracterização dos participantes. Como os participantes são especializados em diferentes áreas do conhecimento, isso pode desbalancear o grupo de tarefas desse estudo. Para tentar mitigar isso, desenvolvemos uma heurística baseada no algoritmo de *round-robin* para redirecionar e dividir os participantes de forma homogênea. No entanto, não podemos garantir que todos os participantes responderam fielmente as perguntas do questionário de caracterização. Dessa forma, respostas que não condizem com o perfil real do participante podem criar grupos desbalanceados. A maioria dos participantes do estudo são alunos de graduação, o que limita a representatividade das pessoas neste estudo. Temos um total de 42 participantes voluntários neste estudo experimental. Embora seja um número pequeno, 30 participantes são considerados uma amostra suficientemente grande para um experimento controlado com pessoas [45]. Assim, esse número de amostras aumenta a validade estatística das conclusões obtidas.

O participante recebe instruções antes de iniciar qualquer uma das fases do experimento. Para impedir que os participantes alterem as respostas já preenchidas, bloqueamos o acesso aos formulários anteriores. Nesse caso, o participante pode apenas avançar nas questões dos formulários, e nunca retroceder. Essa é uma ação necessária, pois retroceder às tarefas já respondidas pode gerar resultados imprecisos e com um grande número de *outliers*, impossibilitando a avaliação da pesquisa. O cronômetro em cada tarefa é oculto.

Decidimos não exibi-lo para evitar que os participantes se sintam pressionados pelo tempo ao realizar as tarefas. Nesse caso, não podemos garantir que tais ameaças foram eliminadas do nosso estudo, porém, as precauções para minimizá-las foram implementadas e avaliadas anteriormente por meio de testes piloto.

Para finalizar, como este estudo foi realizado online, não podemos monitorar ou avaliar os participantes até a submissão das tarefas do experimento. Então, os participantes podem obter vantagens nesse tipo de tarefa utilizando consultas externas, o que pode ser uma ameaça significativa a este estudo. Minimizamos esta ameaça inserindo limites de tempo para cada uma das tarefas (apesar do limite não ter sido atingido por nenhum participante em nenhuma tarefa). Além disso, os códigos-fonte fornecidos nas tarefas foram inseridos em formato de imagem para dificultar que o participante copie os códigos e os executem em máquinas locais para responder as questões mais rapidamente. Vale mencionar que o design das tarefas foi criado para evitar que participantes tentem acessar qualquer material de apoio pela Internet.

## 5.6 Considerações Finais

Este capítulo apresentou os resultados obtidos e as repostas das questões de pesquisa deste estudo. Após validação dos dados e avaliação estatística dos resultados, o ProspectiveProv se mostrou mais eficaz e eficiente quando o comparamos com o ProvenanceCurious. Em comparação com o Visustin, não há diferença significativa em relação a eficácia. Em termos de eficiência, o Visustin teve melhores resultados apenas na tarefa  $A_1$  (mais simples) quando comparado ao ProspectiveProv. Porém, o grande ponto negativo do Visustin é não verificar a corretude do código.

Comparando o uso do ProspectiveProv e os scripts, não há diferença significativa em termos de eficácia e eficiência. Segundo os participantes de computação, 100% deles preferem utilizar *scripts*. Porém, este cenário muda quando avaliamos os participantes de outras áreas, que responderam (60% deles) que preferem utilizar os diagramas como forma de analisar códigos.

# Capítulo 6

## Conclusão

Um dos principais pontos do uso de proveniência está em descrever processos complexos de forma abstrata, sendo um caminho para ajudar os cientistas a identificar informações relevantes que expliquem como um experimento gerou um determinado resultado. No contexto de tarefas computacionais, a proveniência é muito importante na questão de controle de um experimento, análise de dados, reproduzibilidade e identificação de dependências. Nesse quesito, muitas abordagens são dedicadas a coletar a proveniência de *scripts*, tais como YesWorkflow [42], noWorkflow [44, 53], No+Yesworkflow [15], ProvenanceCurious [32, 31], entre outros.

No entanto, apesar dessas abordagens coletarem a proveniência, muitas delas não geram diagramas intuitivos, o que pode dificultar a análise da proveniência. Além disso, a maior parte das abordagens mencionadas se preocupa em gerar grafos de proveniência prospectiva ou retrospectiva, sem relacionar ambos os tipos de proveniência em apenas um grafo. Para cobrir essas lacunas, esta dissertação propõe o ProspectiveProv, uma abordagem destinada a gerar diagramas de proveniência prospectiva de scripts Python. O ProspectiveProv utiliza dados de proveniência capturados pelo noWorkflow para gerar visualizações que combinam a proveniência retrospectiva (relacionada a execução do código) com a prospectiva (o que se refere a estrutura do *script*).

### 6.1 Resultados

Conforme apresentado no Capítulo 3, as abordagens existentes para gerar diagramas de proveniência de scripts ainda possuem limitações, principalmente na questão de unir a proveniência prospectiva e retrospectiva. Esta dissertação apresenta alguns resultados para aumentar o apoio a pesquisa na área de proveniência de scripts, fornecendo alguns

pontos relevantes, tais como:

- Os diagramas do ProspectiveProv se mostraram mais eficientes e eficazes quando comparados às notações de proveniência prospectiva geradas pelo ProvenanceCurious.
- O ProspectiveProv possibilita unir proveniência prospectiva e retrospectiva em uma única notação. Desse modo, é possível entender visualmente partes da estrutura de um script que não foram ativadas e também o conteúdo das variáveis naquele ponto. Abordagens parecidas com essa não permitem realizar isso de maneira automática, dependendo de anotações na estrutura do código, como é o caso do No+YesWorkflow[15].
- Nos resultados experimentais, o ProspectiveProv se mostrou tão eficaz quanto o Visustin, e tão eficiente quanto ele em tarefas mais complexas (tarefas  $A_2$  e  $A_3$ ). No entanto, cabe ressaltar que o Visustin possui uma limitação em relação à garantia da corretude do código, pois pelo fato dele não executar previamente os *scripts*, não existe uma validação ou identificação de erros de sintaxe. Portanto, os diagramas gerados pelo Visustin não garantem que a estrutura do script é válida e funcional.
- Em comparativo entre os diagramas gerados pelo ProspectiveProv e os *scripts*, os resultados mostraram que ambas as abordagens são igualmente eficazes e eficientes. No entanto, participantes de áreas não relacionadas à computação mostraram preferência pelos diagramas do ProspectiveProv.

## 6.2 Limitações

Com base nos resultados apresentados nessa dissertação, é importante mencionar algumas limitações observadas tanto relacionadas ao estudo experimental quanto à abordagem ProspectiveProv. Dessa forma, deve-se mencionar que mesmo tendo um número significante estatisticamente de participantes, o perfil de participantes de computação pode beneficiar algumas avaliações do experimento, como é o caso do estudo comparativo entre diagramas e scripts (uma vez que a maioria dos participantes declarou preferir usar os scripts ao invés dos diagramas). Além disso, o grau de experiência elevado dos participantes com relação a Python pode influenciar o tempo de resposta, já que alguns deles podem não ter contato direto com diagramas de código.

Com relação à arquitetura do ProspectiveProv, uma limitação está relacionada à quantidade de nós que podem representar um script. Nessa circunstância, muitos dos testes foram realizados com experimentos relativamente pequenos. Alguns testes com scripts mais complexos podem gerar um número muito grande de nós, gerando um problema para analisar visualmente a proveniência. Para tentar mitigar esse problema, é possível gerar diagramas menores no ProspectiveProv limitando a linha inicial e final do código. Embora isso sendo possível, essa limitação ainda é evidente e pretendemos tratá-la em trabalhos futuros.

## 6.3 Trabalhos Futuros

Com base na abordagem proposta nesse trabalho, é possível descrever novos temas de pesquisa a partir do tema apresentado. Nesse ponto, as principais propostas para a realização de trabalhos futuros são modificações e melhorias relacionadas aos métodos e heurísticas para gerar grafos de proveniência. Além dessas, novos experimentos também são essenciais para avaliarmos como a abordagem utilizada neste trabalho pode auxiliar cientistas de outras áreas que não Ciência da Computação.

Mesmo que os diagramas possam ser intuitivos, o grande número de nós gerados em grafos mais complexos pode dificultar a análise de dados. Assim, uma sugestão para trabalhos futuros é a realização de novos estudos para aprimorar os métodos de visualização de dados de proveniência. Nesse contexto, uma possibilidade é desenvolver diagramas de proveniência interativos e simplificados. Desse modo, tais modificações permitem destacar partes relevantes de um experimento usando apenas diagramas de proveniência, além de possibilitar que os cientistas interajam diretamente com o diagrama de proveniência sem a necessidade de modificar os experimentos de *scripts*. Simplificar a forma como os cientistas interagem com seus experimentos computacionais é relevante para otimizar o tempo para novas descobertas.

Outra proposta está relacionada a utilizar os próprios diagramas de proveniência como forma de avaliar otimizações na estrutura de um código e modificá-lo sem a necessidade de interagir com os códigos. Adicionalmente, tais modificações podem possibilitar uma nova forma de como os pesquisadores manipulam e analisam os dados de experimentos executados como *scripts*.

# Referências

- [1] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., KUDLUR, M., LEVENBERG, J., MONGA, R., MOORE, S., MURRAY, D., STEINER, B., TUCKER, P., VASUDEVAN, V., WARDEN, P., WICKE, M., YU, Y., ZHANG, X. Tensorflow: A system for large-scale machine learning. In *OSDI* (2016).
- [2] ABBOTT, B., ET AL, R. Gw151226: Observation of gravitational waves from a 22-solar-mass binary black hole coalescence. *Physical review letters* 116 (2016).
- [3] ACAR, U., BUNEMAN, P., CHENEY, J., VAN DEN BUSSCHE, J., KWASNICKOWSKA, N., VANSUMMEREN, S. A graph model of data and workflow provenance. In *Proceedings of the 2nd Conference on Theory and Practice of Provenance* (USA, 2010), TAPP'10, USENIX Association, p. 8.
- [4] AKOUSH, S., SOHAN, R., HOPPER, A. Hadoopprov: Towards provenance as a first class citizen in mapreduce. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance* (USA, 2013), TaPP '13, USENIX Association.
- [5] ALTINTAS, I., BERKLEY, C., JAEGER, E., JONES, M., LUDÄSCHER, B., MOCK, S. Kepler: an extensible system for design and execution of scientific workflows. *Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004.* (2004), 423–424.
- [6] AMANN, B., CONSTANTIN, C., CARON, C., GIROUX, P. Weblab prov: computing fine-grained provenance links for xml artifacts. In *EDBT '13* (2013).
- [7] ATMANSCHACHER, H., LAMBERT, L. B., FOLKERS, G., SCHUBIGER, P. Relevance relations for the concept of reproducibility. *Journal of the Royal Society Interface* 11 (2014).
- [8] BERGSTRA, J., BASTIEN, F., TURIAN, J., PASCANU, R., DELALLEAU, O., BREULEUX, O., LAMBLIN, P., DESJARDINS, G., ERHAN, D., BENGIO, Y., OPÉRATIONNELLE, D. D. E. R. Deep learning on gpus with theano. In *In The Learning Workshop* (2010).
- [9] BUNEMAN, P., KHANNA, S., TAN, W. Why and where: A characterization of data provenance. In *Database Theory - ICDT* (Berlin, Heidelberg, 2001), p. 316–330.
- [10] CALLAHAN, S. P., FREIRE, J., SANTOS, E., SCHEIDECKER, C. E., SILVA, C. T., VO, H. T. Vistrails: Visualization meets data management. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2006), SIGMOD '06, Association for Computing Machinery, p. 745–747.

- [11] CASADEVALL, A., FANG, F. Reproducible science. *Infection and Immunity* 78 (2010), 4972–4975.
- [12] CASS, S. The 2019 top programming languages. *IEEE Spectrum* (2019).
- [13] CREWS, T., ZIEGLER, U. The flowchart interpreter for introductory programming courses. *FIE '98. 28th Annual Frontiers in Education Conference. Moving from 'Teacher-Centered' to 'Learner-Centered' Education. Conference Proceedings (Cat. No.98CH36214)* 1 (1998), 307–312 vol.1.
- [14] DAVISON, A. Automated Capture of Experiment Context for Easier Reproducibility in Computational Research. *Computing in Science Engineering* 14, 4 (2012), 48–56.
- [15] DEY, S., BELHAJJAME, K., KOOP, D., RAUL, M., LUDÄSCHER, B. Linking prospective and retrospective provenance in scripts. In *7th USENIX Workshop on the Theory and Practice of Provenance (TaPP 15)* (Edinburgh, Scotland, 2015), USENIX Association.
- [16] D.J., F. The Fisher-Yates test of significance in 2 x 2 contingency tables. *Biometrika* 35, 1/2 (1948), 145–156.
- [17] DOL, S. Fe.g.: An animated flowchart with example to teach the algorithm based courses in engineering. *2015 IEEE Seventh International Conference on Technology for Education (T4E)* (2015), 49–52.
- [18] DONOHO, D. L. An invitation to reproducible computational research. *Biostatistics* 11, 3 (07 2010), 385–388.
- [19] EINSTEIN, A. *Relativity: The Special and the General Theory*, 1st edition ed. Methuen & Co Ltd, London, 1920.
- [20] ELLKVIST, T., KOOP, D., ANDERSON, E. W., FREIRE, J., SILVA, C. T. Using provenance to support real-time collaborative design of workflows. In *IPAW* (2008).
- [21] ELLSON, J., GANSNER, E. R., KOUTSOFIOS, E., NORTH, S. C., WOODHULL, G. Graphviz and dynagraph – static and dynamic graph drawing tools. *AT&T Labs - Research* (2003).
- [22] FREIRE, J., BONNET, P., SHASHA, D. Computational reproducibility: State-of-the-art, challenges, and database research opportunities. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2012), SIGMOD '12, Association for Computing Machinery, p. 593–596.
- [23] FREIRE, J., CHIRIGATI, F. Provenance and the different flavors of computational reproducibility. *IEEE Data Engineering Bulletin* 41 (2018), 15–26.
- [24] FREIRE, J., KOOP, D., SANTOS, E., SILVA, C. Provenance for computational tasks: A survey. *Computing in Science & Engineering* 10, 3 (2008), 11–21.
- [25] GILBERT, J. E. A. Gsmodutils: a python based framework for test-driven genome scale metabolic model development. *Bioinformatics (Oxford, England)* 35 (2019).

- [26] GLAVIC, B., ESMALI, K. S., FISCHER, P., TATBUL, N. Ariadne: managing fine-grained provenance on data streams. In *DEBS '13* (2013).
- [27] GOECKS, J., NEKRUTENKO, A., TAYLOR, J. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology* 11 (2010), R86.
- [28] GREENWOOD, M., GOBLE, C., STEVENS, R., ZHAO, J., ADDIS, M., MARVIN, D., MOREAU, L., OINN, T., WATSON, P. Provenance of e-science experiments - experience from bioinformatics. In *Proceedings of UK e-Science All Hands Meeting 2003* (2003), p. 223–226.
- [29] HERSCHEL, M., DIESTELKÄMPER, R., LAHMAR, H. B. A survey on provenance: What for? what form? what from? *The VLDB Journal* 26 (2017), 881–906.
- [30] HULL, D., WOLSTENCROFT, K., STEVENS, R., GOBLE, C., POCOCK, M., LI, P., OINN, T. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research* 34 (2006), W729 – W732.
- [31] HUQ, M. R. *An inference-based framework for managing data provenance*. Ph.D. Dissertation. University of Twente, 2001.
- [32] HUQ, M. R., APERS, P. M. G., WOMBACHER, A. ProvenanceCurious: a tool to infer data provenance from scripts. In *EDBT* (2013), p. 765–768.
- [33] INTERLANDI, M., SHAH, K., TETALI, S. D., GULZAR, M., YOO, S., KIM, M., MILLSTEIN, T., CONDIE, T. Titian: Data provenance support in spark. *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases* 9 (2015), 216 – 227.
- [34] JANDRE, E., DIIRR, B., BRAGANHOLO, V. Provenance in collaborative in silico scientific research: A survey. *SIGMOD Rec.* 49, 2 (dezembro de 2020), 36–51.
- [35] KEIDING, N. Reproducible research and the substantive context. *Biostatistics* 11 3 (2010), 376–8.
- [36] KINNEAR, T. C., TAYLOR, J. R. *Marketing Research: An Applied Approach*. McGraw-Hill, 1983.
- [37] LERNER, B., BOOSE, E. RDataTracker: collecting provenance in an interactive scripting environment. *TAPP* (2014), 1–4.
- [38] LIM, C., LU, S., CHEBOTKO, A., FOTOUHI, F. Prospective and retrospective provenance collection in scientific workflow environments. *2010 IEEE International Conference on Services Computing* (2010), 449–456.
- [39] MARTA-ALMEIDA, M., RUIZ-VILLARREAL, M., OTERO, P., COBAS, M., PELIZ, A., NOLASCO, R., CIRANO, M., PEREIRA, J. Oofε: A python engine for automating regional and coastal ocean forecasts. *Environ. Model. Softw.* 26 (2011), 680–682.
- [40] MATTOSO, M., WERNER, C., TRAVASSOS, G., BRAGANHOLO, V., OGASAWARA, E., DE OLIVEIRA, D., DA CRUZ, S. M. S., MARTINHO, W., MURTA, L. Towards supporting the life cycle of large scale scientific experiments. *Int. J. Bus. Process. Integr. Manag.* 5 (2010), 79–92.

- [41] MAY, D., TAMURA, K., NOBLE, W. S. Detecting modifications in proteomics experiments with param-medic. *Journal of proteome research* 18 4 (2019), 1902–1906.
- [42] MCPHILLIPS, T., SONG, T., KOLISNIK, T., AULENBACH, S., BELHAJJAME, K., BOCINSKY, K., CAO, Y., CHIRIGATI, F., DEY, S., FREIRE, J., OTHERS. YesWorkflow: A User-Oriented, Language-Independent Tool for Recovering Workflow Information from Scripts. *International Journal of Digital Curation* (2015).
- [43] MISSIER, P., LUDASCHER, B., BOWERS, S., DEY, S., SARKAR, A., SHRESTHA, B., ALTINTAS, I., ANAND, M. K., GOBLE, C. Linking multiple workflow provenance traces for interoperable collaborative science. *The 5th Workshop on Workflows in Support of Large-Scale Science* (2010), 1–8.
- [44] MURTA, L., BRAGANHOLO, V., CHIRIGATI, F., KOOP, D., FREIRE, J. noworkflow: Capturing and analyzing provenance of scripts. In *IPAW* (2014), p. 1–12.
- [45] N. JURISTO, A. M. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Spain, 2001.
- [46] OY, C. A. Visustin flowchart generator, 1997.
- [47] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., LOUPPE, G., PRETTENHOFER, P., WEISS, R., WEISS, R. J., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., DUCHESNAY, E. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.* 12 (2011), 2825–2830.
- [48] PENG, R. Reproducible research in computational science. *Science* 334 (2011), 1226 – 1227.
- [49] PIMENTEL, J. A. F. *Provenance from Scripts*. Tese de Doutorado, Niterói, RJ, 2021.
- [50] PIMENTEL, J. F., FREIRE, J., BRAGANHOLO, V., MURTA, L. Tracking and analyzing the evolution of provenance from scripts. In *Provenance and Annotation of Data and Processes - 6th International Provenance and Annotation Workshop, IPAW 2016, Proceedings* (2016), B. Glavic and M. Mattoso, Eds., Lecture Notes in Computer Science, Springer Verlag, p. 16–28.
- [51] PIMENTEL, J. F., FREIRE, J., MURTA, L., BRAGANHOLO, V. A survey on collecting, managing, and analyzing provenance from scripts. *ACM Comput. Surv.* 52, 3 (junho de 2019).
- [52] PIMENTEL, J. F., MISSIER, P., MURTA, L., BRAGANHOLO, V. Versioned-PROV: A PROV extension to support mutable data entities. In *Procs. IPAW 2018* (London, 2018), Springer.
- [53] PIMENTEL, J. F., MURTA, L., BRAGANHOLO, V., FREIRE, J. noworkflow: a tool for collecting, analyzing, and managing provenance from python scripts. *Proc. VLDB Endow.* 10 (2017), 1841–1844.

- [54] ROSENZWEIG, I., HODGES, B. A python wrapper for coupling hydrodynamic and oil spill models. *Technical (Online) Report - University of Texas at Austin, Center for Research in Water Resources* 11 (2011), 1–48.
- [55] RYAN K DALE, BRENT PEDERSEN, A. Q. Pybedtools: A flexible python library for manipulating genomic datasets and annotations. *Bioinformatics (Oxford, England)* (2011).
- [56] SÁENZ, J., ZUBILLAGA, J., FERNÁNDEZ, J. Geophysical data analysis using python. *Computers & Geosciences* 28 (2002), 457–465.
- [57] SAMUEL, S., KÖNIG-RIES, B. Provenance oriented reproducibility of scripts using the reproduce-me ontology. In *The Semantic Web: ESWC 2017 Satellite Events* (2017), Springer International Publishing, p. 17–20.
- [58] SAMUEL, S., TAUBERT, F., WALTHER, D., KÖNIG-RIES, B., BÜCKER, H. Towards reproducibility of microscopy experiments. *D-Lib Magazine* 23 (2017).
- [59] SANDVE, G. K., NEKRUTENKO, A., TAYLOR, J., HOVIG, E. Ten simple rules for reproducible computational research. *PLOS Computational Biology* 9 (2013), 1–4.
- [60] SCANLAN, D. A. Structured flowcharts outperform pseudocode: an experimental comparison. *IEEE Software* 6 (1989), 28–36.
- [61] SIMMHAN, Y., PLAIE, B., GANNON, D. A survey of data provenance in e-science. *SIGMOD Record* 34 (2005), 31–36.
- [62] SROKA, J., WŁODARCZYK, P., KRUPA, L., HIDDERS, J. Dfl designer: collection-oriented scientific workflows with petri nets and nested relational calculus. *Proceedings of the ACM SIGMOD International Conference on Management of Data* (2010).
- [63] TRAVASSOS, G., BARROS, M. Contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering contributions of in virtuo and in silico experiments for the future of empirical studies in software engineering. *The Future of Empirical Studies in Software Engineering: Proceedings of the ESEIW 2003 Workshop on Empirical Studies in Software Engineering, WSESE 2003; Roman Castles, Italy, September 29th, 2003* 2 (01 2003), 117.
- [64] WEINGART, P. The relation between science and technology — a sociological explanation. *The Dynamics of Science and Technology* (01 1978), 251–286.
- [65] XINO GALOS, S. Using flowchart-based programming environments for simplifying programming and software engineering processes. *2013 IEEE Global Engineering Education Conference (EDUCON)* (2013), 1313–1322.

# APÊNDICE A - Mapeamento do Experimento (Fase 1)

Tabela A.1: Lista de diagramas recebidos pelos participantes conforme o esquema de quadrados Latinos.

Grupo	Fase 1		
	Tarefa A1	Tarefa A2	Tarefa A3
$X_1$	Diagrama(A)	Diagrama(B)	Diagrama(C)
$X_2$	Diagrama(F)	Diagrama(D)	Diagrama(E)
$X_3$	Diagrama(H)	Diagrama(I)	Diagrama(G)

Tabela A.2: Detalhe do que se pede em cada uma das questões

Tarefa	Fase 1	
	Questão	
A1	ESCREVA o código Python correspondente ao diagrama a SEGUIR.	
A2	ESCREVA o código Python correspondente ao diagrama a SEGUIR.	
A3	ESCREVA o código Python correspondente ao diagrama a SEGUIR.	

## APÊNDICE B – Mapeamento do Experimento (Fase 2)

Tabela B.1: Lista de diagramas recebidos pelos participantes conforme o esquema de quadrados Latinos.

Grupo	Fase 1	
	Tarefa B1	Tarefa B2
$Y_1$	Script(BY)	Diagrama(AY)
$Y_2$	Diagrama(AY)	Script(BY)

Perguntas apresentadas para os participantes quanto ao Grupo de experimentos  $Y_i$  (comparação entre *scripts* e diagramas):

- **Tarefa B1** - Quando é o DIAGRAMA. Observe atentamente o diagrama baseado em códigos Python a seguir. Escreva qual será o resultado (o campo que você deve analisar está marcado em vermelho) apresentado com base nas entradas a seguir:  
ENTRADAS:  $X = "ab"$ ,  $Y = "Aa"$
- **Tarefa B2** - Quando é o SCRIPT. Observe atentamente o código Python a seguir. Escreva qual será o resultado apresentado por esse código na linha 33 com base nas entradas fornecidas a seguir: ENTRADAS:  $idA = [1, 0, 1]$   $idB = [2, 0, 3]$

## APÊNDICE C – Lista de Imagens (Fase 1)

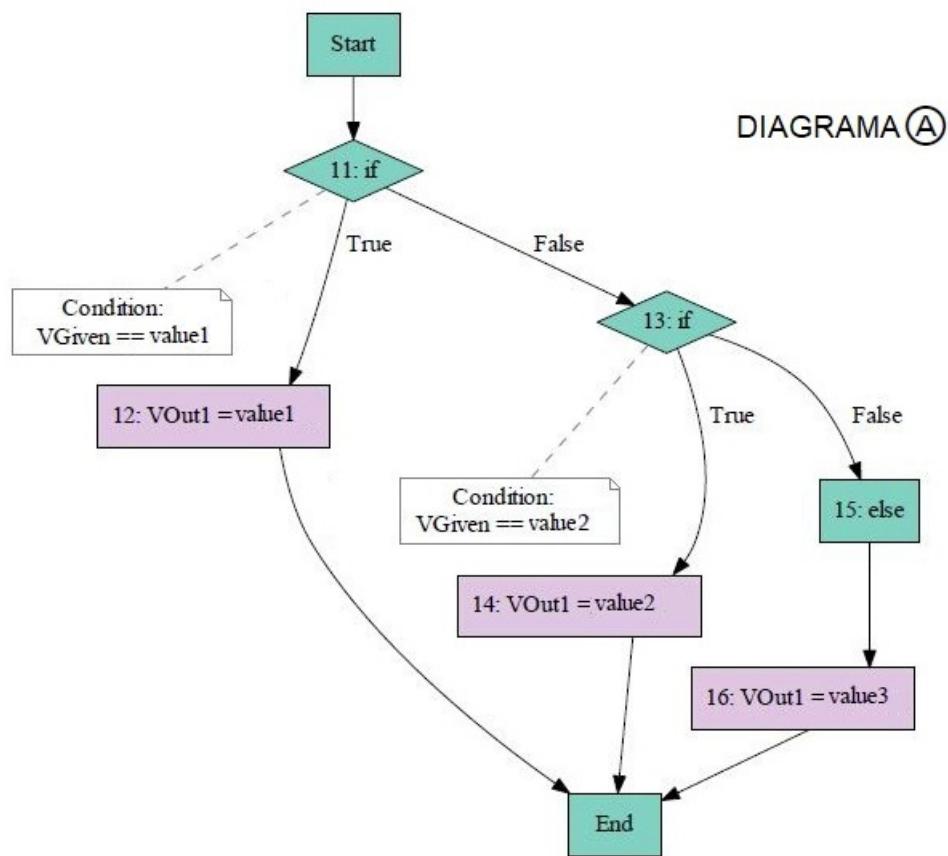


DIAGRAMA B

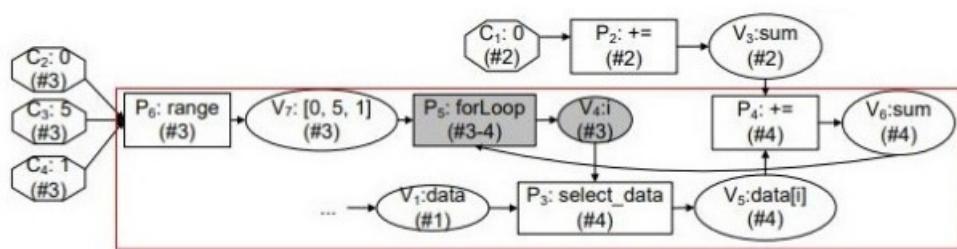
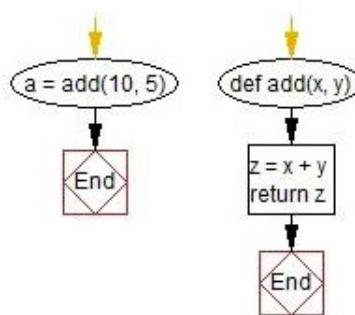


DIAGRAMA C



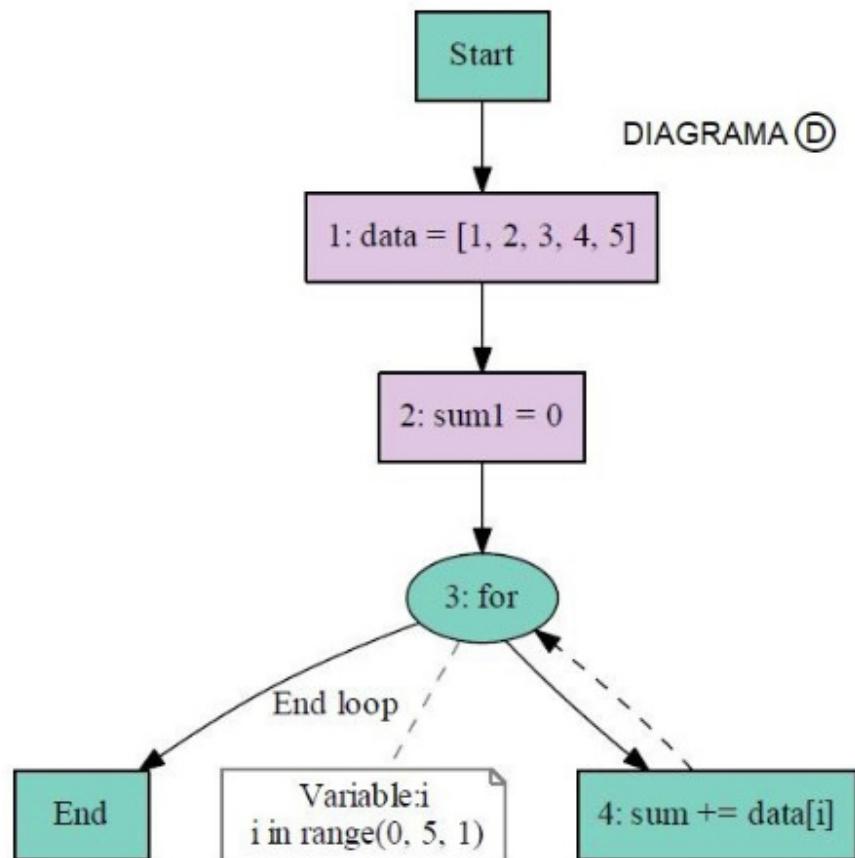
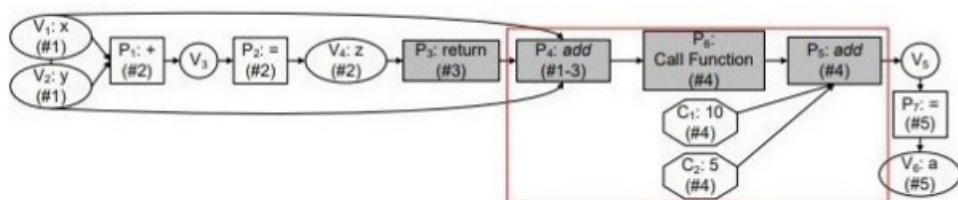
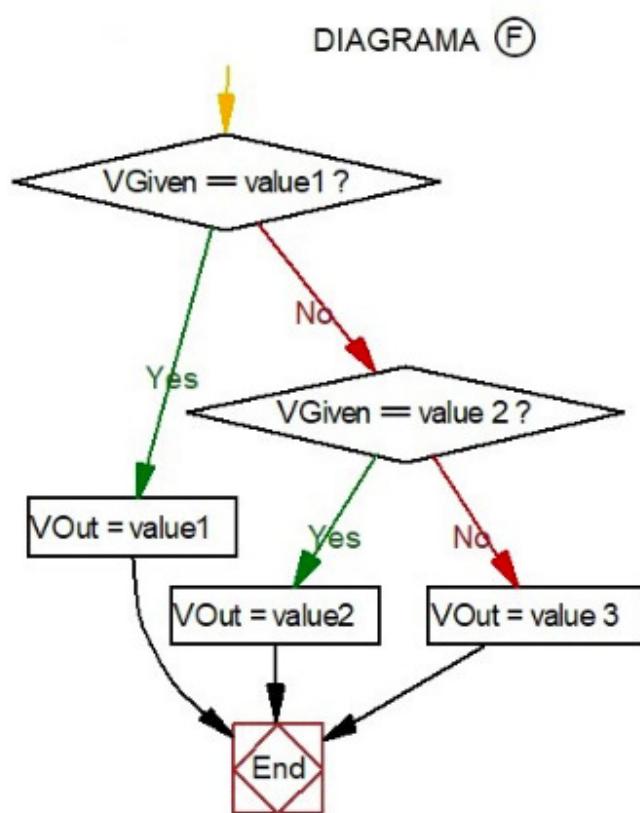


DIAGRAMA E





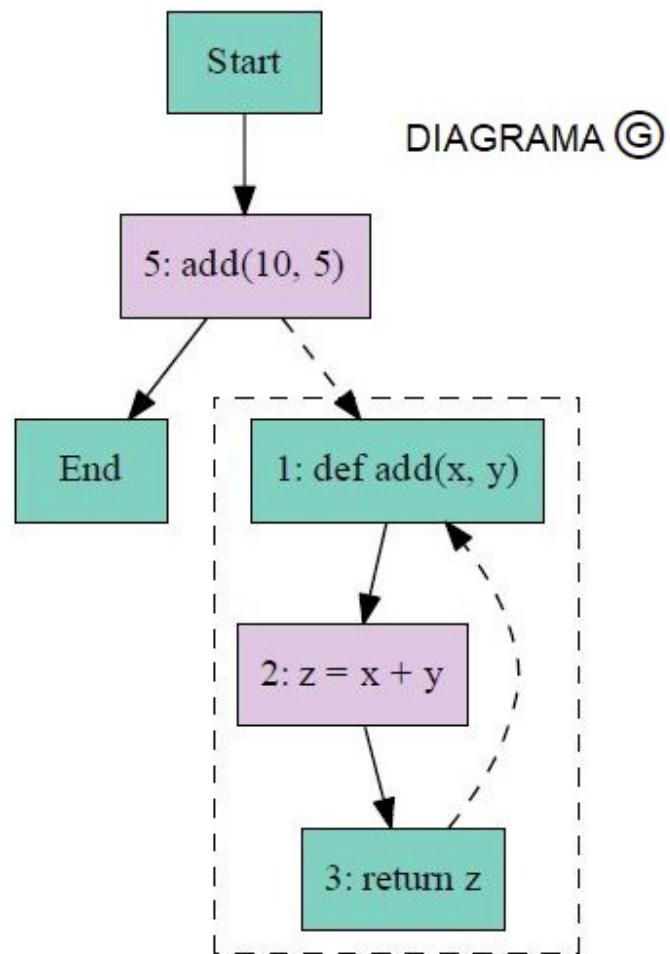
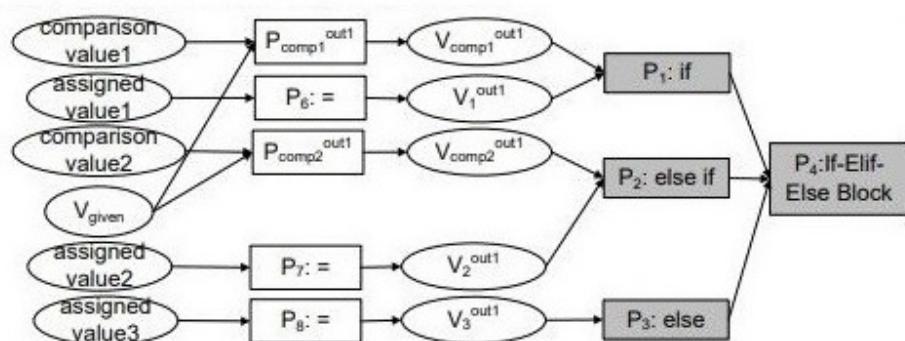
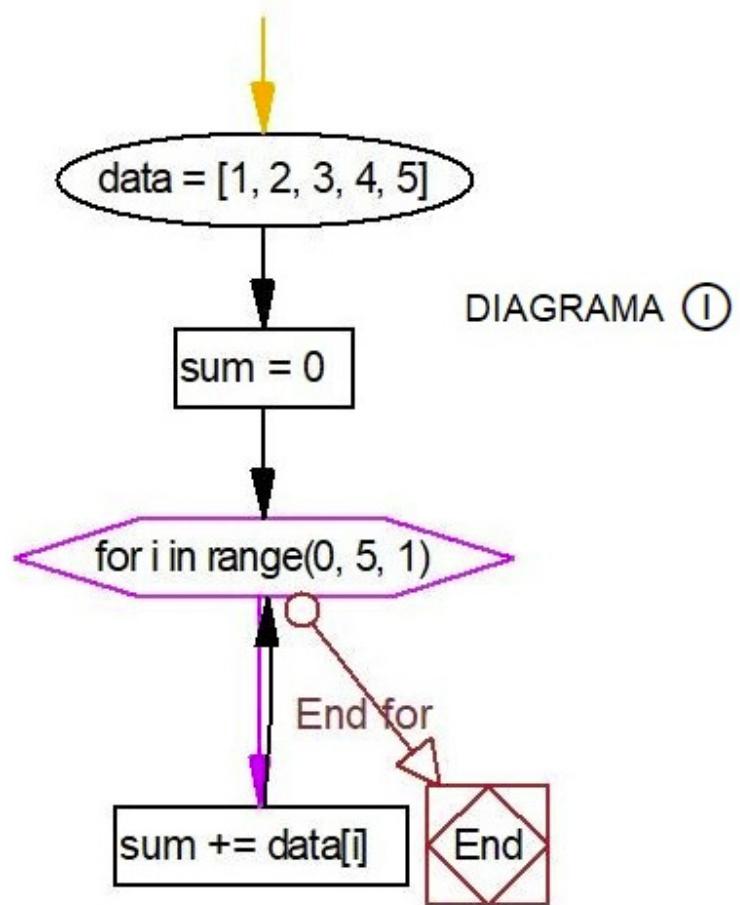


DIAGRAMA H





## APÊNDICE D - Lista de Imagens (Fase 2)

```
01. def crossing(id_A, id_B):
02.     if id_A != id_B:
03.         condicao_1 = id_A >= 6
04.         condicao_2 = id_B < 8
05.
06.         if condicao_1 and condicao_2:
07.             return id_A + 1
08.         else:
09.             if condicao_1:
10.                 return id_A + 2
11.             if condicao_2:
12.                 return id_B
13.         else:
14.             return id_A + 1
15.
16. def mutation(x, y):
17.     i = len(passaporte)
18.     while i != 0:
19.         if i == y:
20.             temp = passaporte[x]
21.             passaporte[x] = passaporte[i]
22.             passaporte[i] = temp
23.
24.         i = i - 1
25.     return passaporte
26.
27. id_A = [1, 0, 1]
28. id_B = [2, 0, 3]
29. passaporte = [0, 0, 0]
30. i = 0
31. while i < len(id_A):
32.     passaporte[i] = crossing(id_A[i], id_B[i])
33.     print('i: ', i, ' passaporte: ', passaporte) ←
34.     i = i + 1
35.
36. resultado = mutation(1, 2)
37. print(resultado)
```

Figura D.1: Script de exemplo utilizado no experimento

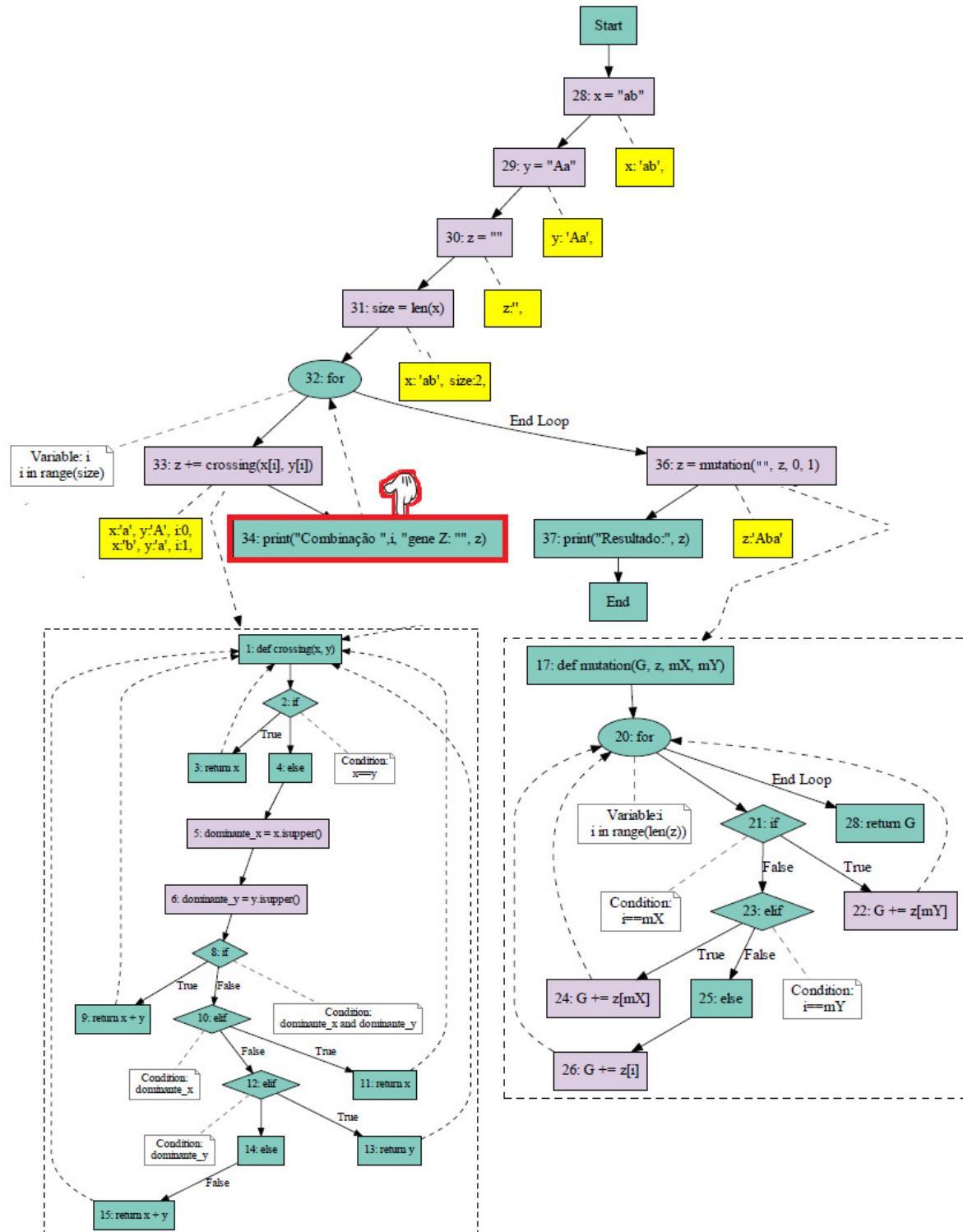


Figura D.2: Diagrama de exemplo utilizado no experimento

## APÊNDICE E – Questionário de Avaliação 1

### Questionário de avaliação I

Na sua opinião, em uma escala de 1 até 5, indique o grau de dificuldade da TAREFA 1 (onde 1 é extremamente fácil, 2 fácil, 3 médio, 4 difícil, 5 extremamente difícil):

1	2	3	4	5
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
<input type="radio"/>				

Na sua opinião, em uma escala de 1 até 5, indique o grau de dificuldade da TAREFA 2 (onde 1 é extremamente fácil, 2 fácil, 3 médio, 4 difícil, 5 extremamente difícil):

1	2	3	4	5
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
<input type="radio"/>				

Na sua opinião, em uma escala de 1 até 5, indique o grau de dificuldade da TAREFA 3 (onde 1 é extremamente fácil, 2 fácil, 3 médio, 4 difícil, 5 extremamente difícil):

1	2	3	4	5
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
<input type="radio"/>				

Back

Next

Figura E.1: Questionário de Avaliação I

# APÊNDICE F – Questionário de Avaliação II

**Questionário de avaliação II**

---

**Na sua opinião, resolver a tarefa do experimento II foi mais fácil usando qual modelo:**

Utilizando o script Python  
 Utilizando o diagrama

**Na sua opinião, em uma escala de 1 até 5, indique o grau de dificuldade da TAREFA utilizando o SCRIPT (onde 1 é extremamente fácil, 2 fácil, 3 médio, 4 difícil, 5 extremamente difícil):**

1      2      3      4      5

---

**Na sua opinião, em uma escala de 1 até 5, indique o grau de dificuldade da TAREFA utilizando o DIAGRAMA (onde 1 é extremamente fácil, 2 fácil, 3 médio, 4 difícil, 5 extremamente difícil):**

1      2      3      4      5

---

---

[Back](#) [Next](#)

Figura F.1: Questionário de Avaliação II

# APÊNDICE G – Termo de Consentimento Livre e Esclarecido (TCLE)

**Condutor do Estudo:** Vitor Gama Lemos

**Pesquisadores Responsáveis:** Profa Vanessa Braganholo e João Felipe Pimentel

**Instituição:** Universidade Federal Fluminense (UFF)

1. **Sobre o estudo.** Os estudos experimentais procuram avaliar uma determinada abordagem. Esse estudo é conduzido por alunos de Pós-graduação em Computação da Universidade Federal Fluminense (IC-UFF) e consiste na avaliação de resultados obtidos por mecanismos utilizados para representar visualmente scripts Python.
2. **Tratamento de possíveis riscos.** Serão tomadas todas as providências durante a coleta de dados de forma a garantir a sua privacidade e seu anonimato.
3. **Benefícios e Custos.** Espera-se que, como resultado deste estudo, você possa aumentar seus conhecimentos, de maneira a contribuir para o aumento da qualidade das atividades com as quais você trabalhe ou possa vir a trabalhar. Este estudo também contribuirá com resultados importantes para a pesquisa de um modo geral. Você não terá nenhum gasto ou ônus com a sua participação no estudo e também não receberá qualquer espécie de reembolso ou gratificação devido à autorização do uso dos dados coletados nesse estudo.
4. **Confidencialidade da Pesquisa.** Seu nome não será identificado de modo algum. Quando os dados forem coletados, seu nome será removido dos mesmos e não será utilizado em nenhum momento durante a análise ou apresentação dos resultados.
5. **Participação.** Sua participação neste estudo é muito importante e voluntária, pois requer a sua aprovação para utilização dos dados coletados. Você tem o direito de não querer participar ou de sair deste estudo a qualquer momento, sem penalidades.

Caso você decida retirar-se do estudo, favor notificar o pesquisador responsável. Você pode solicitar esclarecimento sobre o estudo antes do início da execução das tarefas do experimento.

6. **Declaração de Consentimento.** Declaro que li e estou de acordo com as informações contidas neste documento e que toda linguagem técnica utilizada na descrição deste estudo de pesquisa foi explicada satisfatoriamente, recebendo respostas para todas as minhas dúvidas. Confirmo também que recebi uma cópia deste Termo (TCLE), e comprehendo que sou livre para não autorizar a utilização dos meus dados neste estudo em qualquer momento, sem qualquer penalidade. Declaro ter mais de 18 anos e concordo de espontânea vontade em participar deste estudo.