Merge and Code Review

Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br

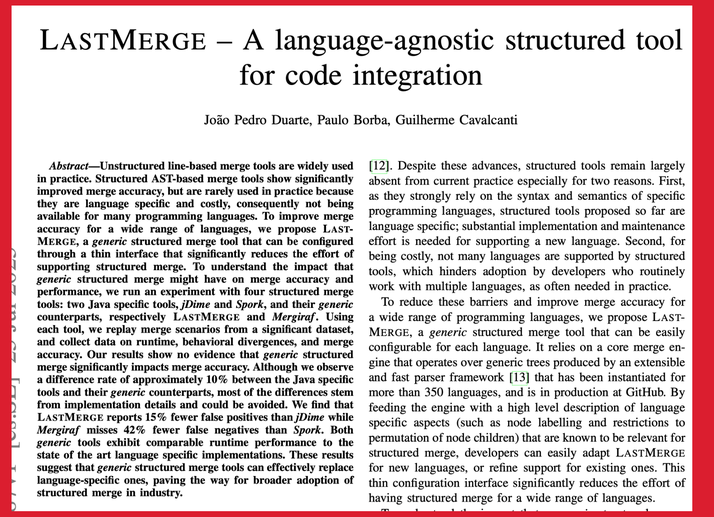
Structured merge

The Impact of Language Independence on Structured Merge Accuracy and Efficiency

João Pedro Henrique Santos Duarte
Paulo Borba
Guilherme Cavalcanti

Centro de Informática UFPE
Universidade Federal de Pernambuco

https://arxiv.org/pdf/2507.19687



LastMerge – A language-agnostic structured tool for code integration

João Pedro Duarte, Paulo Borba, Guilherme Cavalcanti

***Abstract*—Unstructured line-based merge tools are widely used in practice. Structured AST-based merge tools show significantly improved merge accuracy, but are rarely used in practice because they are language specific and costly, consequently not being available for many programming languages. To improve merge accuracy for a wide range of languages, we propose LastMerge, a *generic* structured merge tool that can be configured through a thin interface that significantly reduces the effort of supporting structured merge. To understand the impact that *generic* structured merge might have on merge accuracy and performance, we run an experiment with four structured merge tools: two Java specific tools, *jDime* and *Spork*, and their *generic* counterparts, respectively LastMerge and *Mergiraf*. Using each tool, we replay merge scenarios from a significant dataset, and collect data on runtime, behavioral divergences, and merge accuracy. Our results show no evidence that *generic* structured merge significantly impacts merge accuracy. Although we observe a difference rate of approximately 10% between the Java specific tools and their *generic* counterparts, most of the differences stem from implementation details and could be avoided. We find that LastMerge reports 15% fewer false positives than *jDime* while *Mergiraf* misses 42% fewer false negatives than *Spork*. Both *generic* tools exhibit comparable runtime performance to the state of the art language specific implementations. These results suggest that *generic* structured merge tools can effectively replace language-specific ones, paving the way for broader adoption of structured merge in industry.**

[12]. Despite these advances, structured tools remain largely absent from current practice especially for two reasons. First, as they strongly rely on the syntax and semantics of specific programming languages, structured tools proposed so far are language specific; substantial implementation and maintenance effort is needed for supporting a new language. Second, for being costly, not many languages are supported by structured tools, which hinders adoption by developers who routinely work with multiple languages, as often needed in practice.

To reduce these barriers and improve merge accuracy for a wide range of programming languages, we propose LastMerge, a *generic* structured merge tool that can be easily configurable for each language. It relies on a core merge engine that operates over generic trees produced by an extensible and fast parser framework [13] that has been instantiated for more than 350 languages, and is in production at GitHub. By feeding the engine with a high level description of language specific aspects (such as node labelling and restrictions to permutation of node children) that are known to be relevant for structured merge, developers can easily adapt LastMerge for new languages, or refine support for existing ones. This thin configuration interface significantly reduces the effort of having structured merge for a wide range of languages.

João e Paulo introduzem diferentes modificações em um mesmo método de uma classe

Original

```
public class Account {
  void debit(int amount) {
    // ...
  }
}
```

João

```
public class Account {
  public void debit(int amount) {
    // ...
  }
}
```

Paulo

```
public class Account {
  static void debit(int amount) {
    // ...
  }
}
```

Mas, quando integramos essas modificações...

Merge

```
public class Account {
<<<<<<<
  public void debit(int amount) {
=======
  static void debit(int amount) {
>>>>>>>
    // ...
  }
}
```

Acabamos esbarrando em um **conflito de merge**

Merge

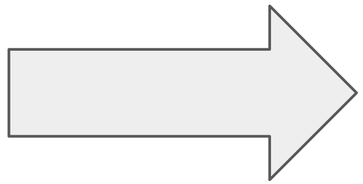
```
public class Account {
<<<<<<<
  public void debit(int amount) {
=======
  static void debit(int amount) {
>>>>>>>
      // ...
  }
}
```

Em Java, modificadores podem ter a ordem permutada

Account.java

```
public class Account {
<<<<<<<
  public void debit(int amount) {
=======
  static void debit(int amount) {
>>>>>>>
    // ...
  }
}
```

Falso positivo

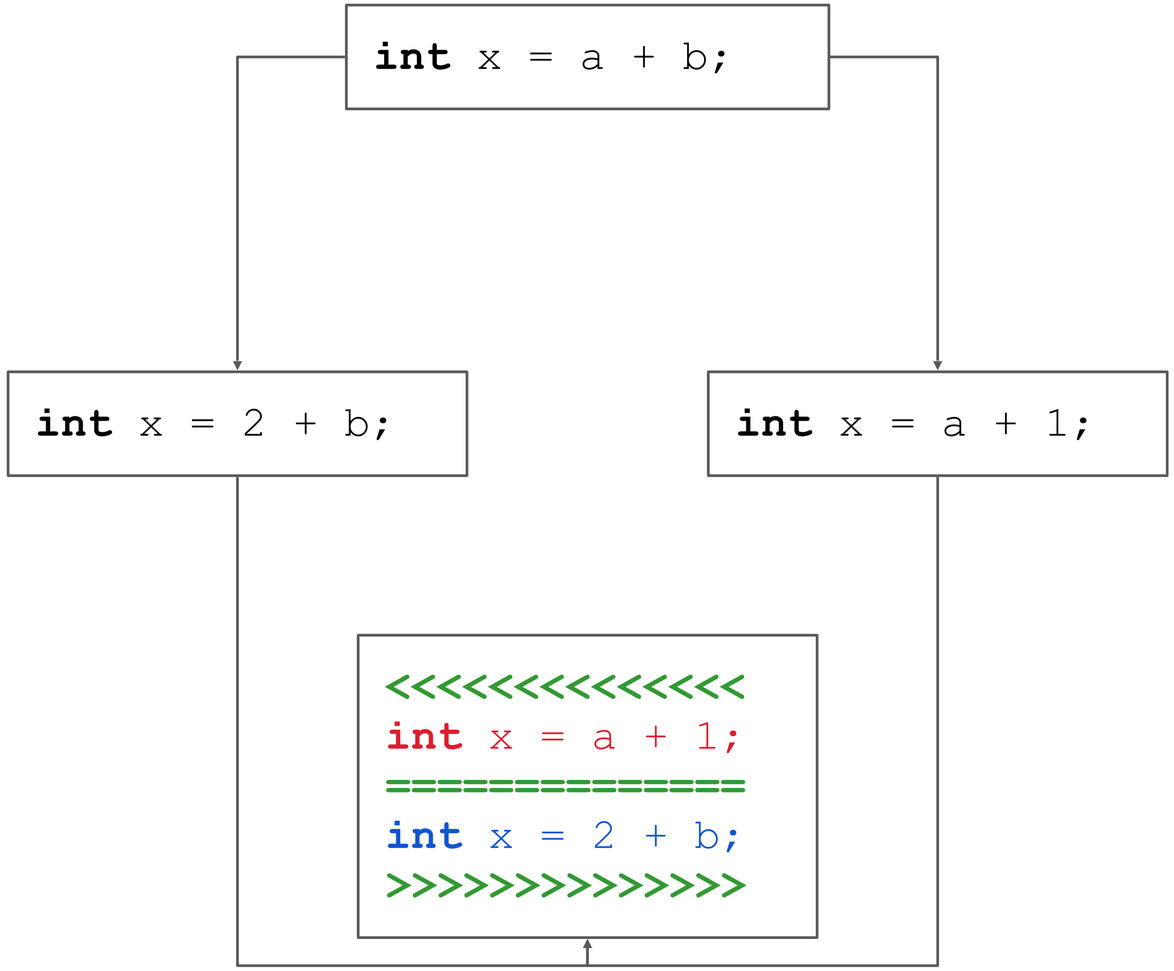


Account.java

```
public class Account {
  public static void debit(int
amount) {
    // ...
  }
}
```

Por que estes falsos positivos acontecem?

Estratégias convencionais consideram apenas a diferença de conteúdo entre as linhas durante o processo de merge.

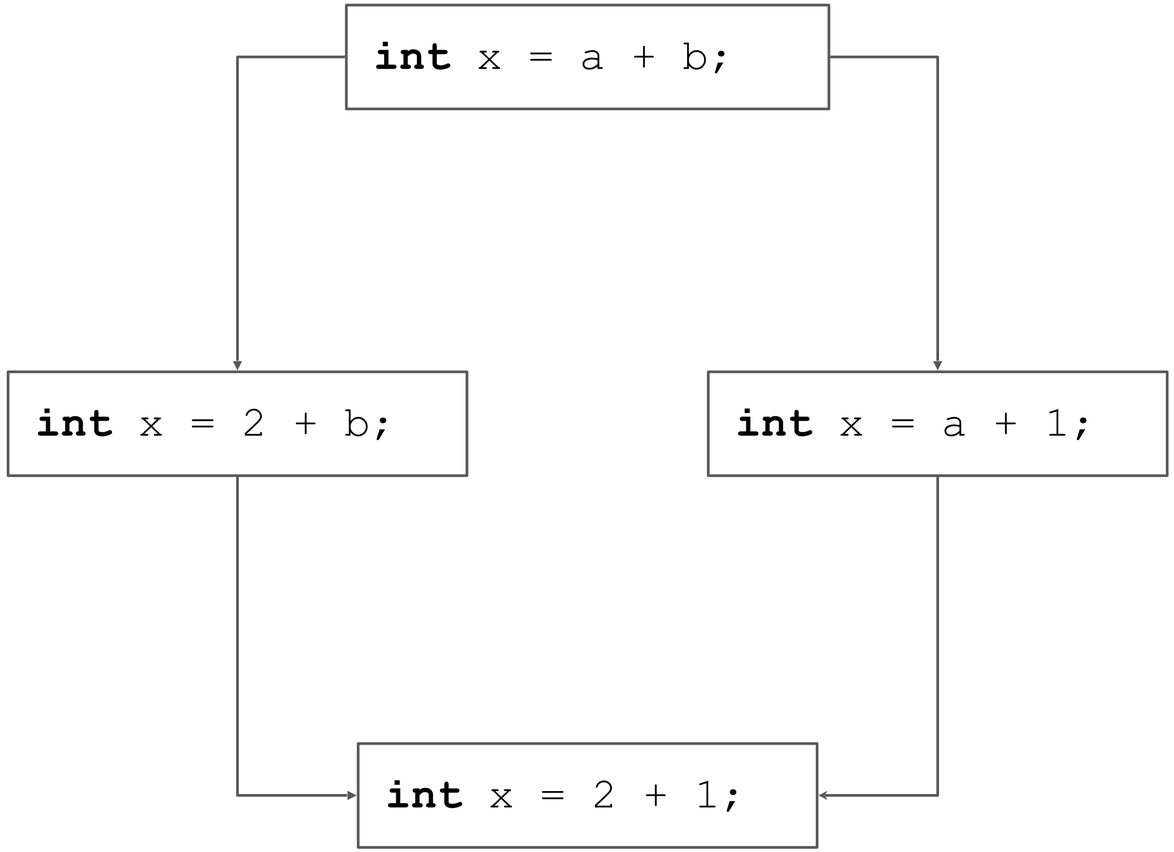


Por que estes falsos positivos acontecem?

Estratégias convencionais consideram apenas a diferença de conteúdo entre as linhas durante o processo de merge.

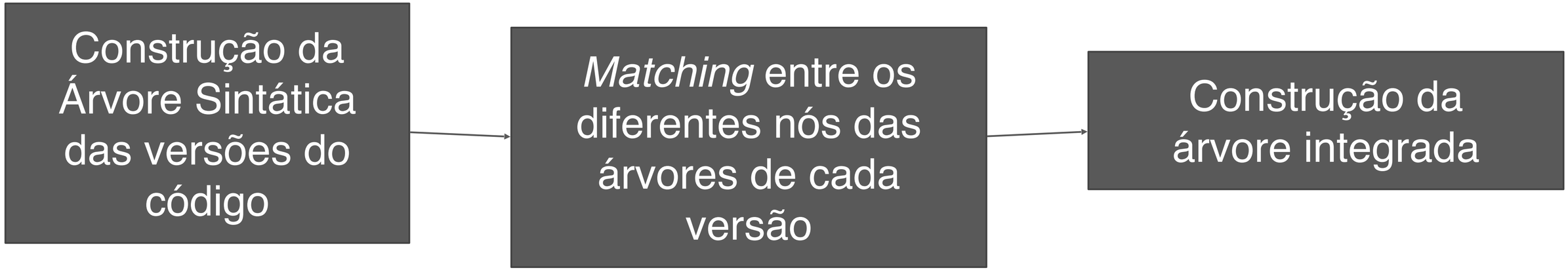
Mas, código não é uma mera sequência de caracteres e linhas.

Tem estrutura bem definida!



Considerando a estrutura do programa

Enxergar os arquivos sendo integrados de maneira mais abstrata.



Ferramentas estruturadas já existem e tem resultados positivos (jDime).

- Redução no número de conflitos reportados em quase 30% [Seibt et al, 2021]

Por conhecer a estrutura da linguagem, merge estruturado resolve conflitos que ferramentas tradicionais não conseguem

João

```
public class Account {
  public void debit(int amount) {
    // ...
  }
}
```

Paulo

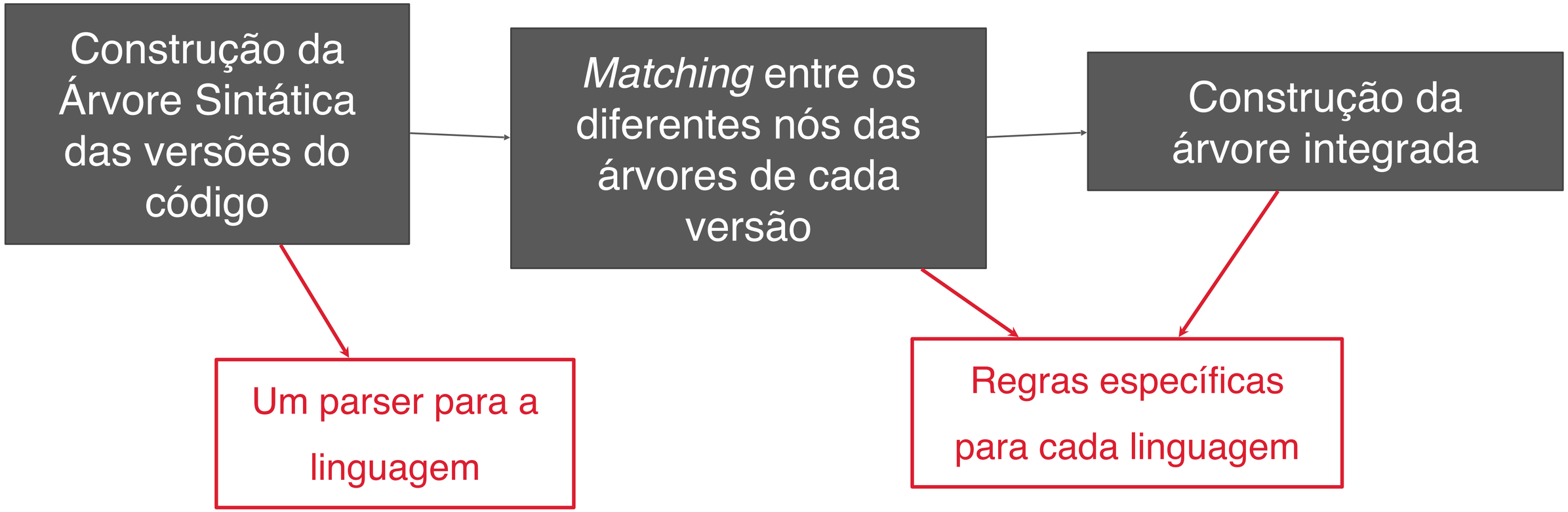
```
public class Account {
  static void debit(int amount) {
    // ...
  }
}
```

Merge

```
public class Account {
  public static void debit(int
amount) {
    // ...
  }
}
```

Então, por que elas não são utilizadas na indústria?

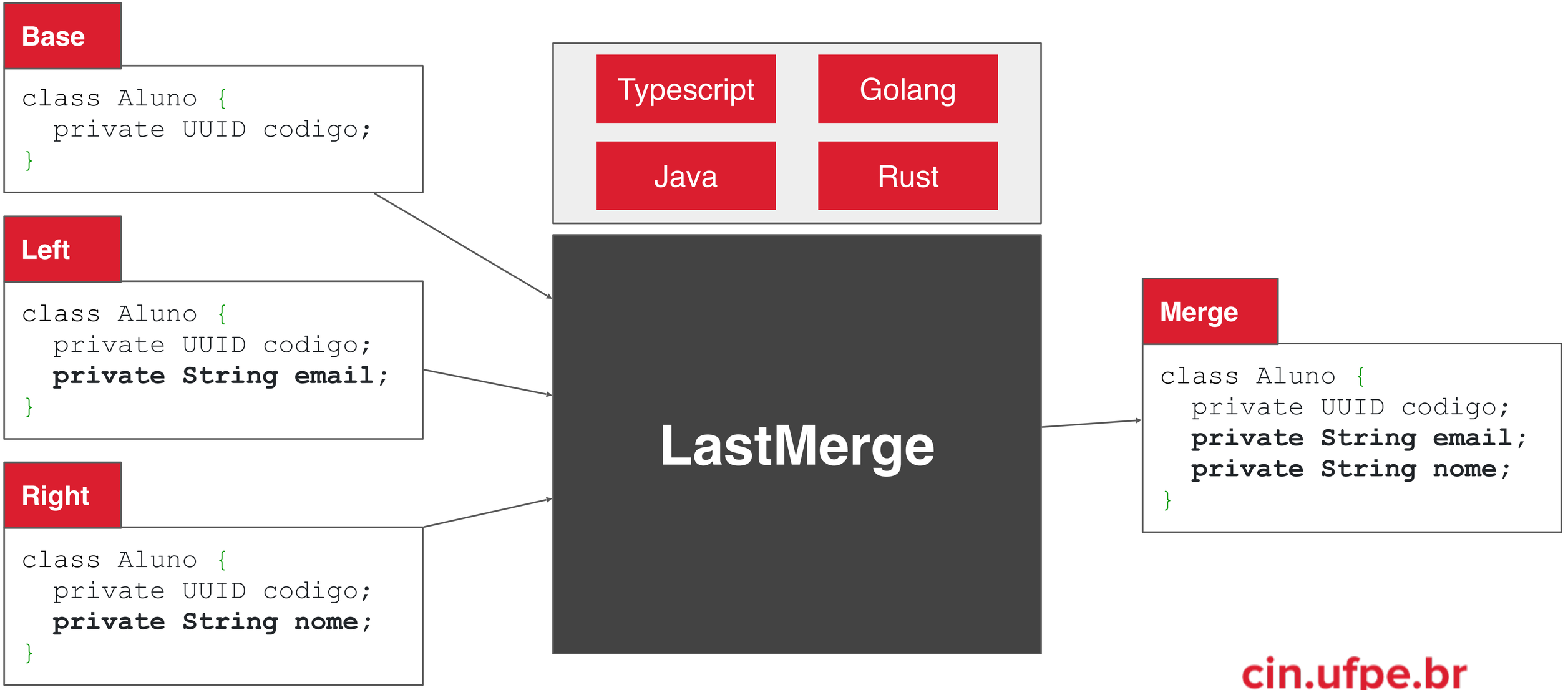
Ferramentas existentes são específicas para uso com uma única linguagem.



E se fosse possível isolar aspectos relacionados à linguagem?

Linguagem de Programação

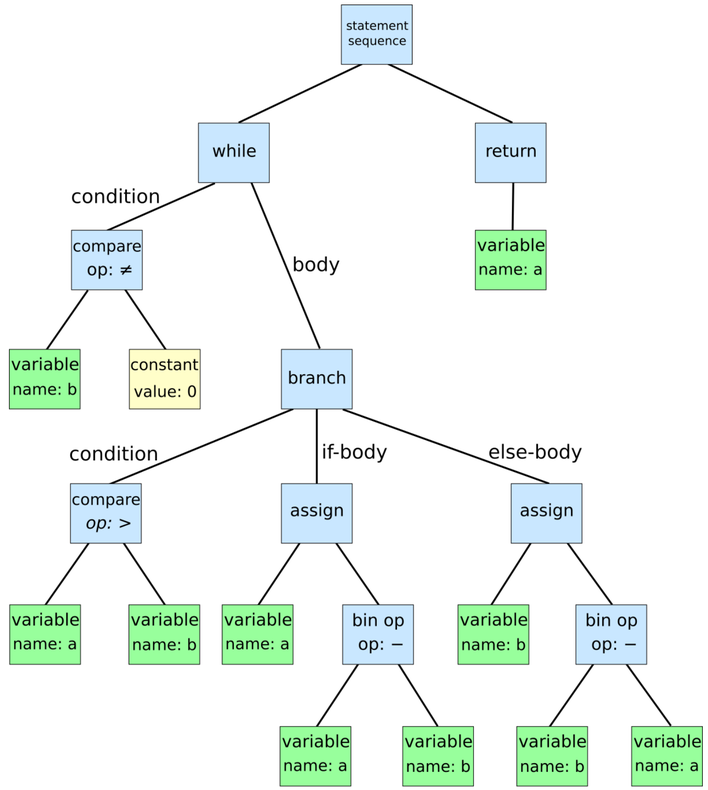
Language Agnostic Structured Tool for Code Merging



Generalizando o *parsing* com Tree Sitter

Código-Fonte

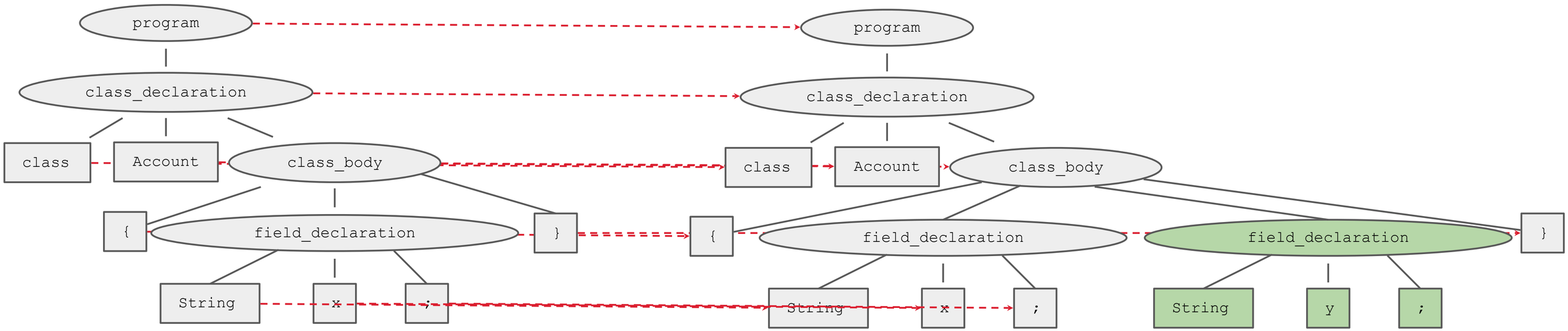
Descrição da Gramática



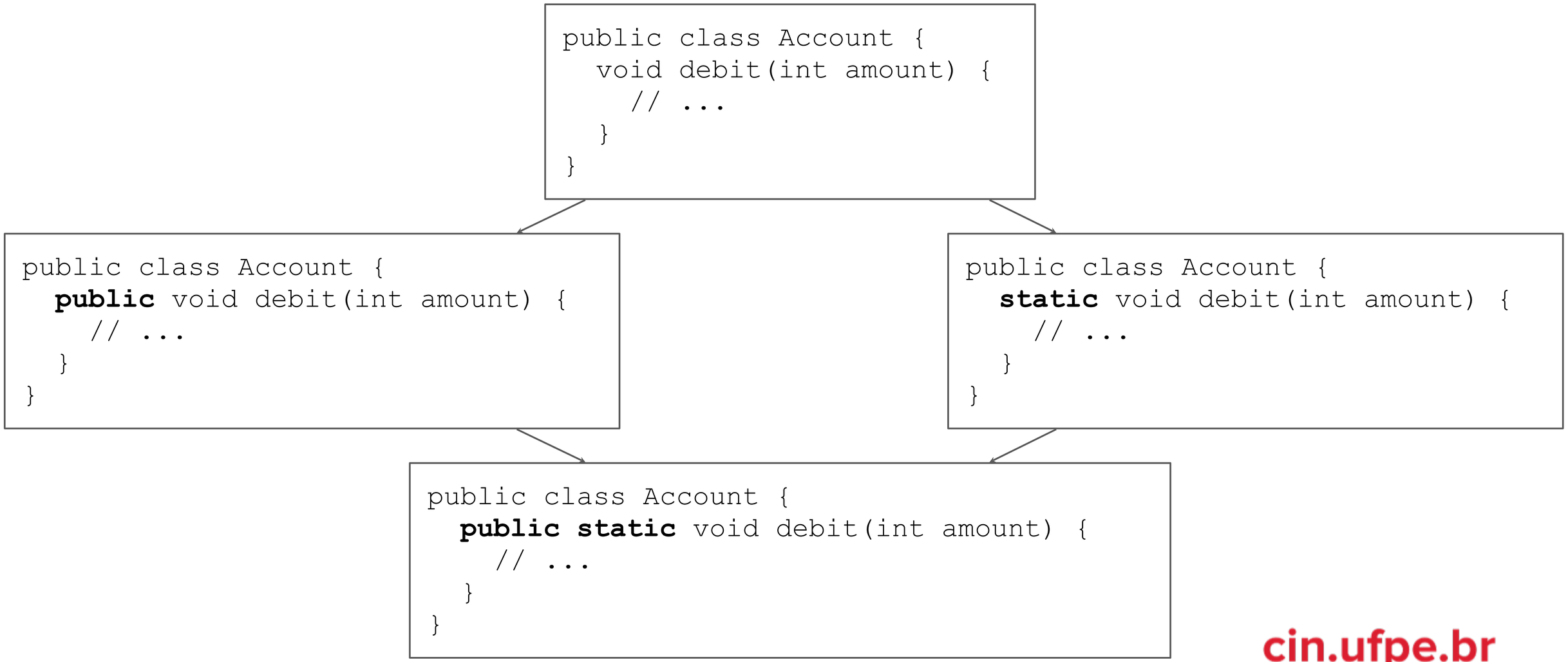
Instanciado para mais de 350 linguagens e usado em produção no GitHub

Matching: encontrando nós correspondentes entre as diferentes árvores

Mesmo algoritmos de jDime; Qual usar depende se podemos reordenar filhos; Level-wise



Merging: usando as informações de matching para inferir as operações realizadas



Mergiraf: outra ferramenta genérica

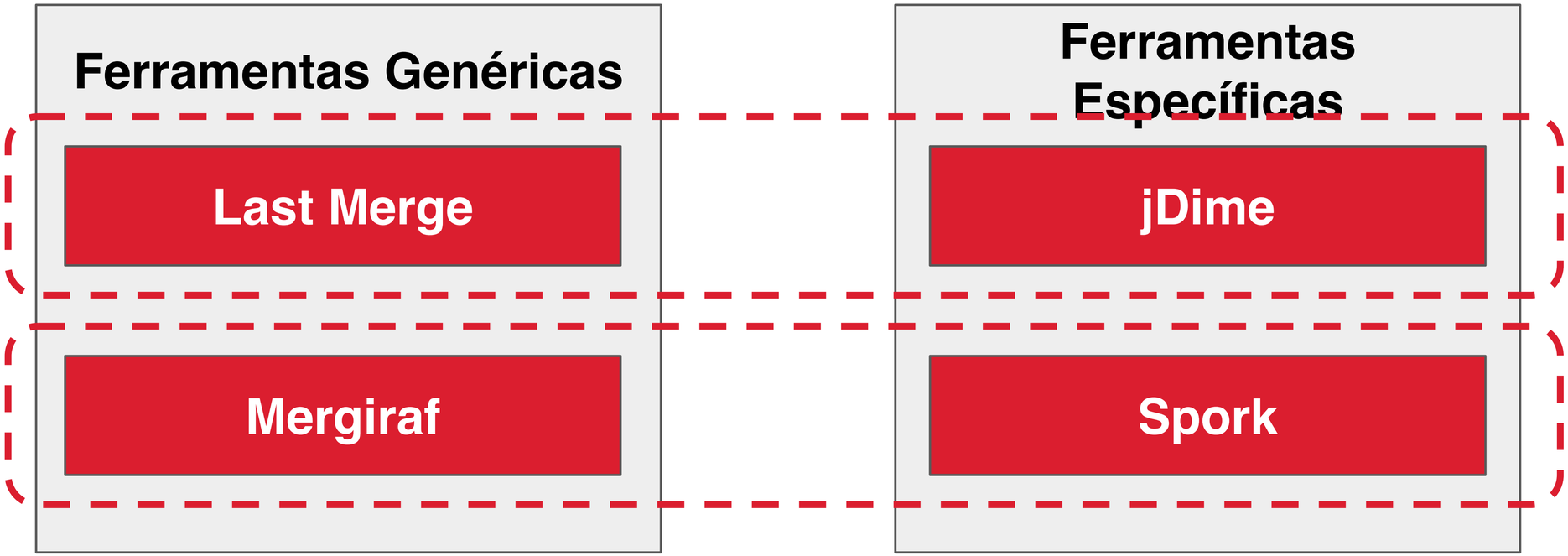
- Também utiliza o Tree Sitter para realizar o parsing do código.
- Reimplementa os algoritmos utilizados pela ferramenta específica Spork
  - GumTree para cálculo dos matchings entre as revisões
  - 3DM-Merge para merge das árvores.

Ferramentas genéricas podem substituir ferramentas específicas?

Estudo comparativo a fim de investigar

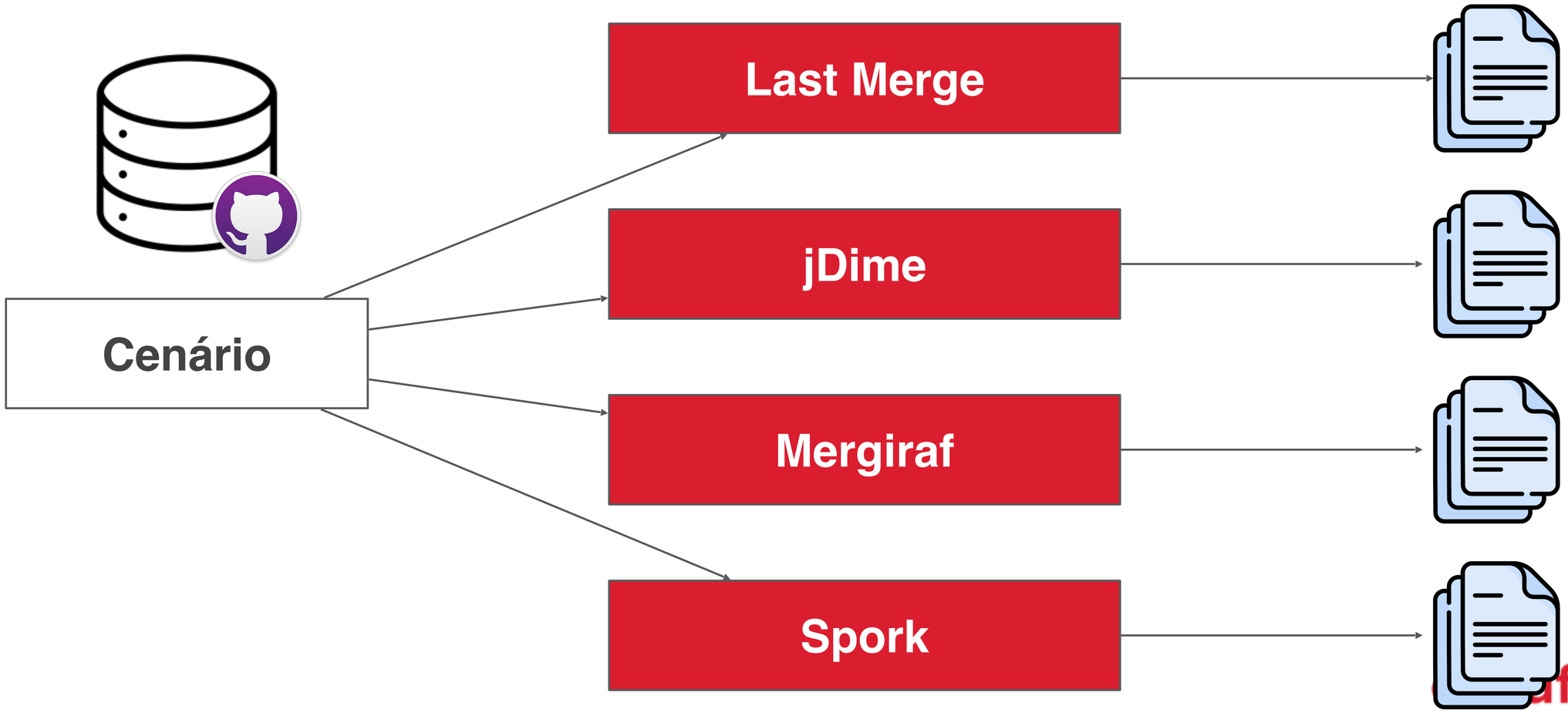
1. Como *merge* estruturado genérico impacta a acurácia?

2. *Merge* estruturado genérico é proibitivo computacionalmente?



Execução de cada ferramenta em cenários oriundos de Schesch et al. (2024)

5.229 cenários, abrangendo 13.675 arquivos; suítes de teste não triviais.



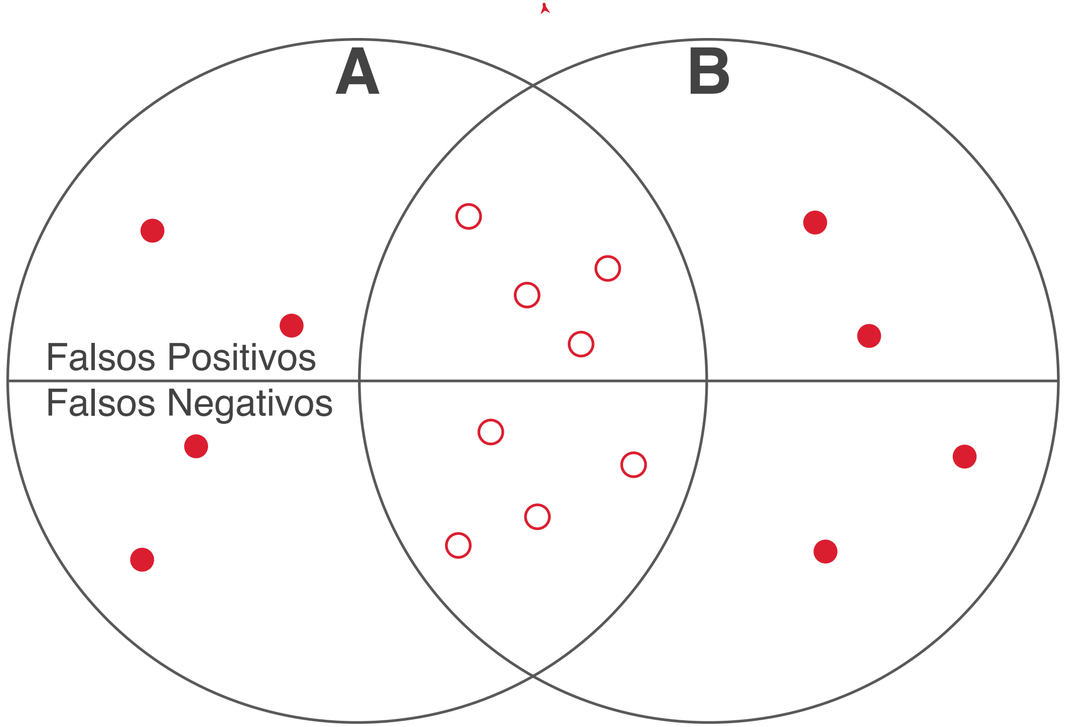
Como saber se a ferramenta integrou o cenário corretamente?

- Um **falso positivo** ocorre quando um conflito é reportado mesmo que, as modificações introduzidas não interfiram entre si;
- Um **falso negativo** ocorre quando as modificações introduzidas interferem entre si e nenhum conflito é reportado pela ferramenta.

Problema: detectar falsos positivos e negativos de maneira absoluta é difícil.

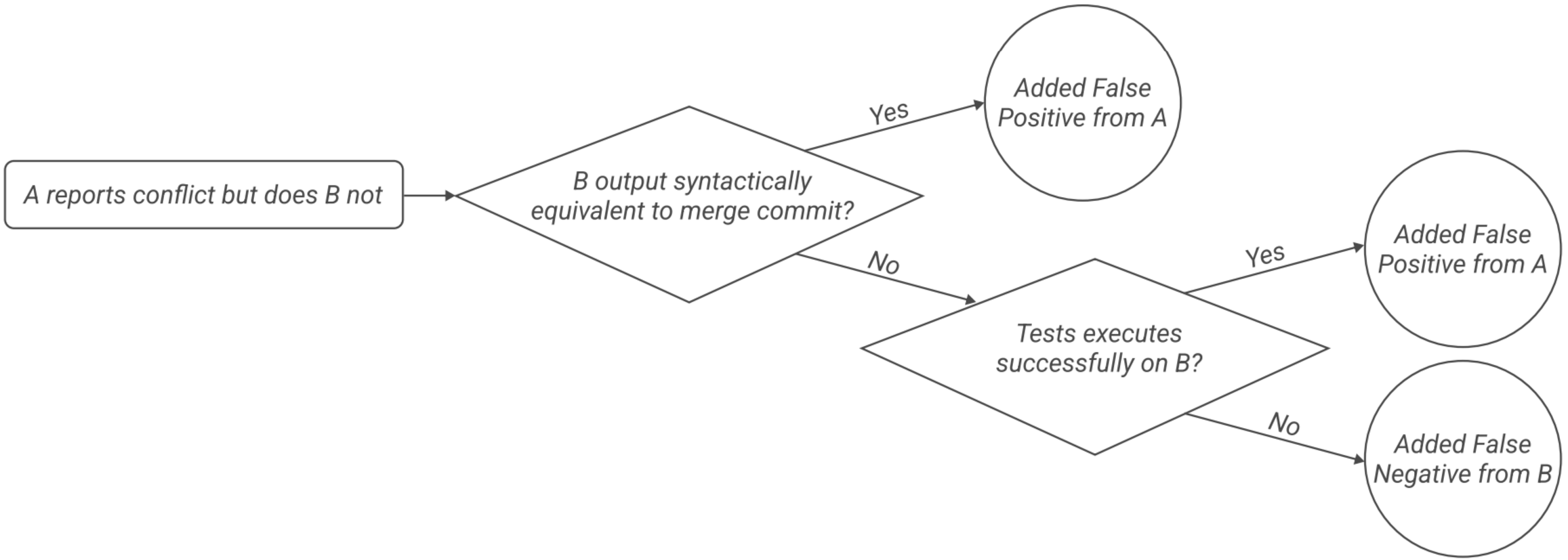
Comparação relativa: onde as ferramentas diferem

Falsos Positivos Adicionados (aFPs)

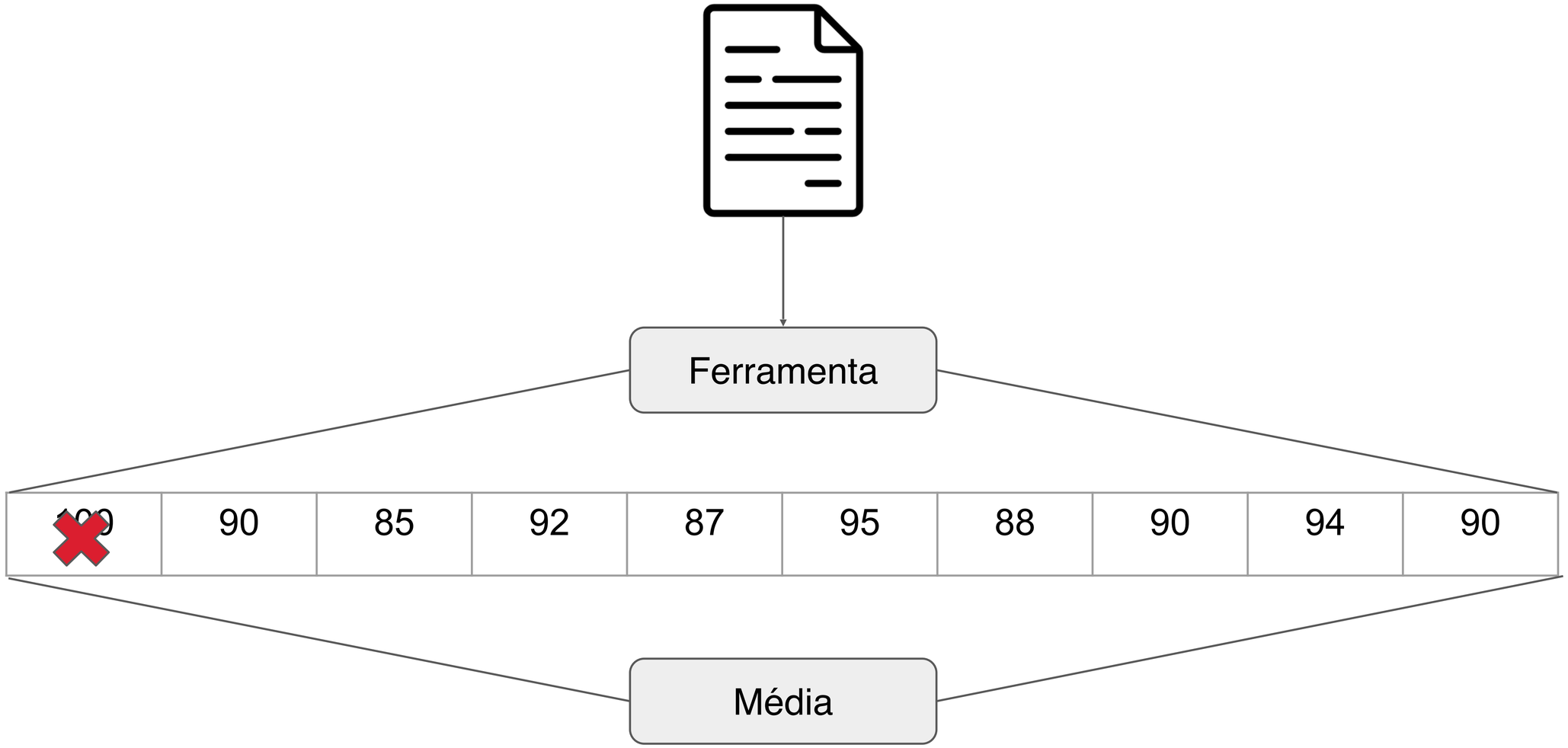


Falsos Negativos Adicionados (aFNs)

Utilizando testes e comparação sintática para identificar aFPs e aFNs



Respondendo RQ2: Computando o tempo de execução de um cenário



Em aproximadamente 10% dos cenários, genérico e específico discordam sobre a existência de conflitos.

Existência de conflitos	jDime vs LastMerge	Spork vs Mergiraf
Concordam	4909 (92,5%)	4697 (88,7%)
Discordam	400 (7,5%)	601 (11,3%)

Como essas diferenças impactam na acurácia do merge?

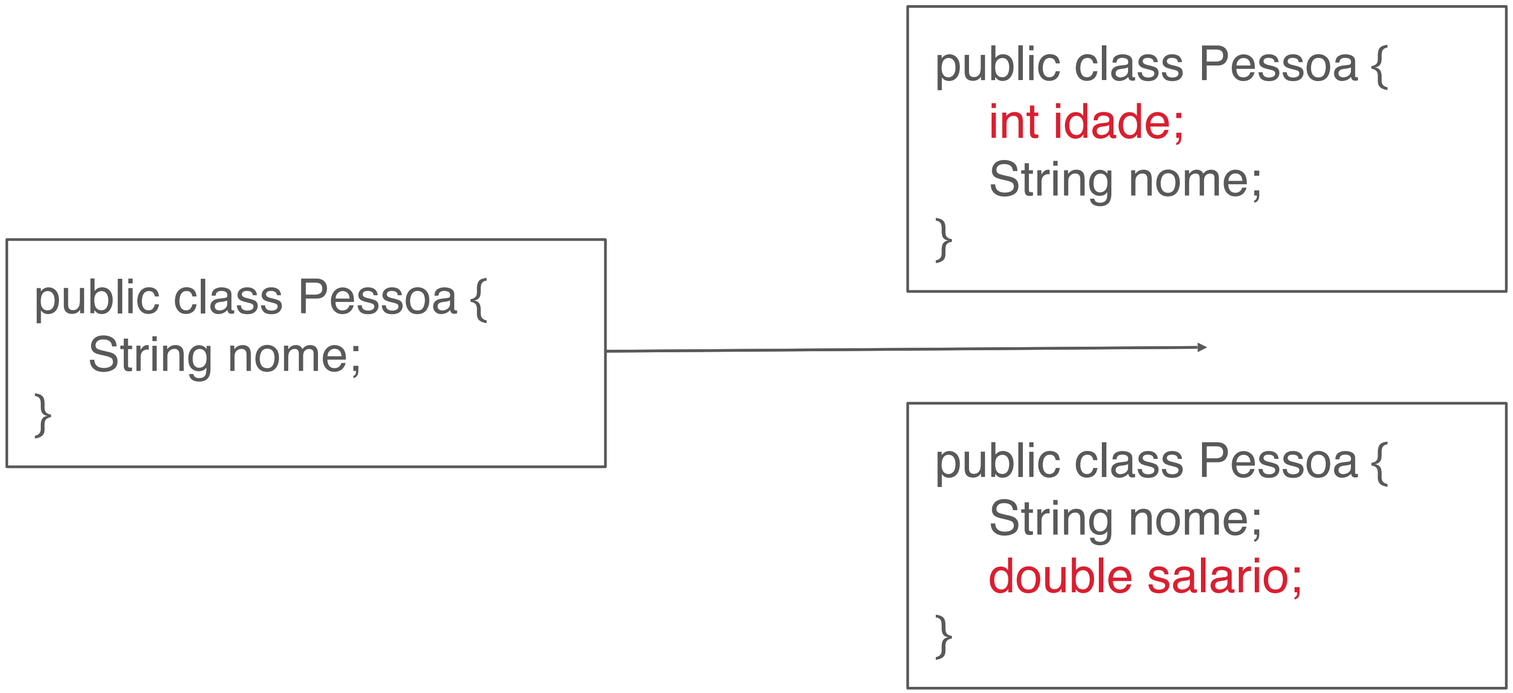
Por que essas diferenças aparecem?

Last Merge possui menos aFPs; jDime, menos aFNs

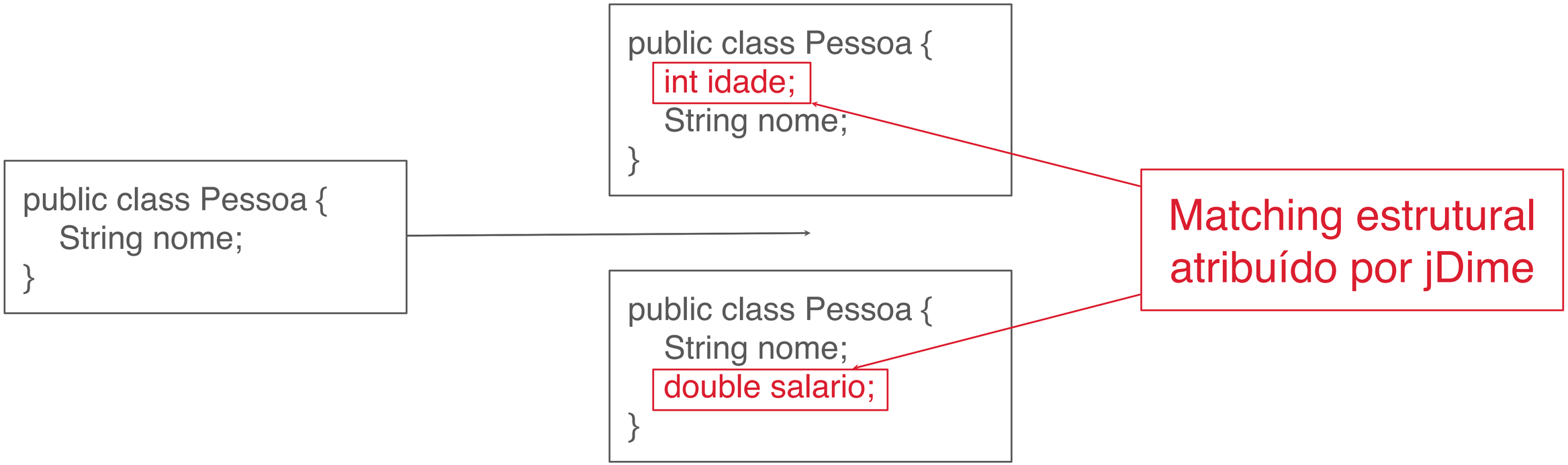
Situação	jDime	LastMerge
Falsos positivos adicionados (aFPs)	153	130
Falsos negativos adicionados (aFNs)	29	85

aFPs de jDime ocorrem por inacurácias na configuração

jDime só atribui identificadores à *imports* e *literais*; outros nós usam matching estrutural

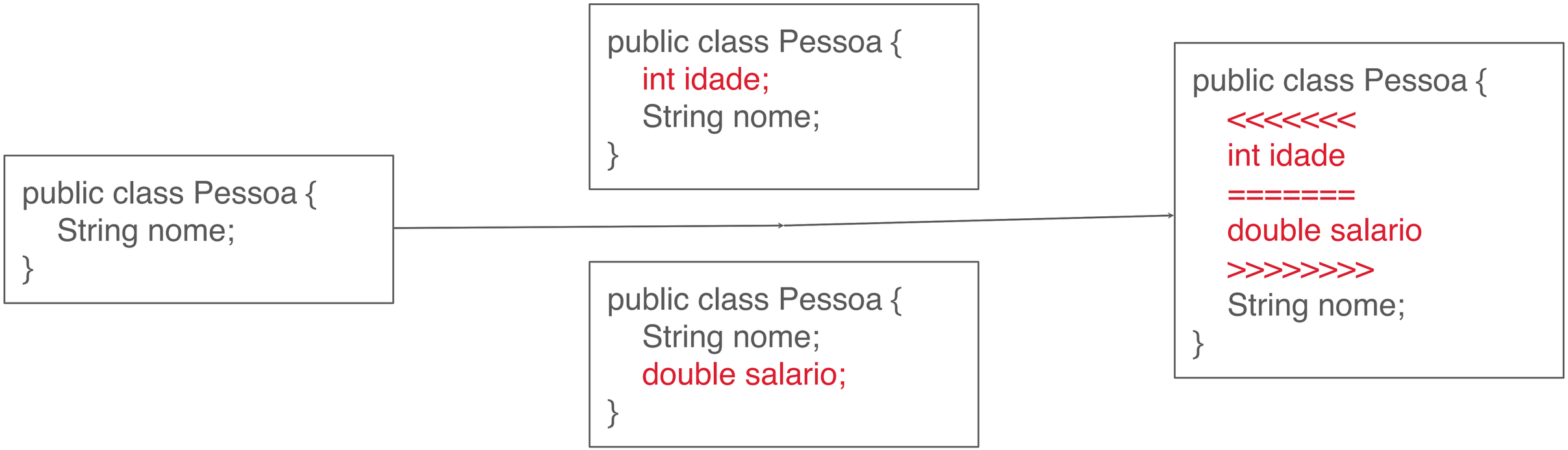


Por não atribuir identificadores, permite fazer o matching entre duas propriedades diferentes



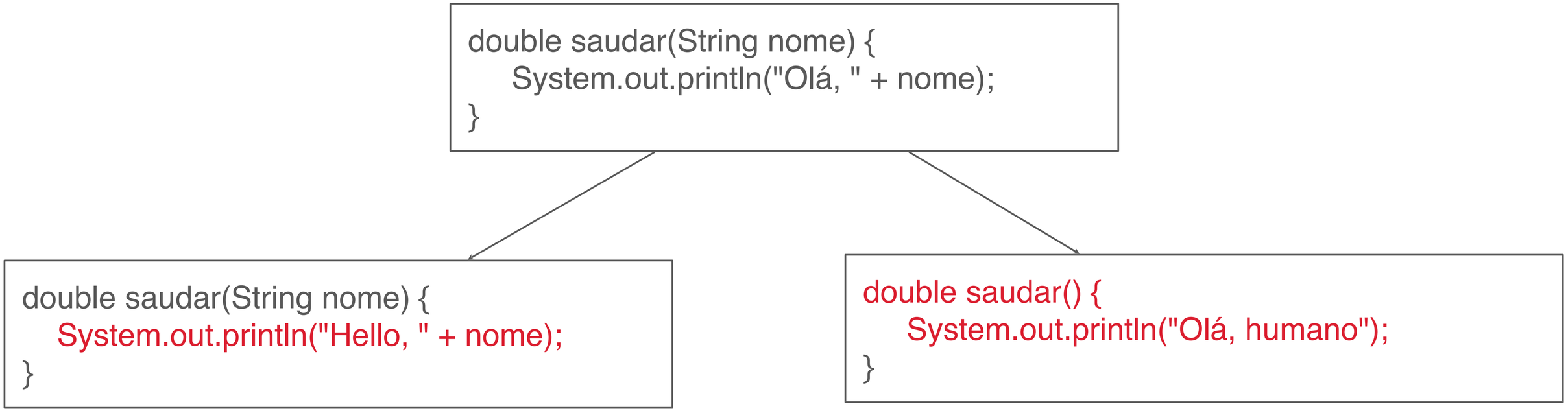
aFPs de jDime ocorrem por inacurácias na configuração

jDime só atribui identificadores a *imports* e *literais*



aFPs de Last Merge são de *renaming* (Cavalcanti et al, 2017)

Uma revisão modifica um método enquanto a outra modifica sua assinatura



Durante o matching, as modificações são incorretamente inferidas

```
double saudar(String nome) {
    System.out.println("Olá, " + nome);
}
```

```
double saudar(String nome) {
    System.out.println("Hello, " + nome);
}
```

```
double saudar() {
    System.out.println("Olá, humano");
}
```

Last Merge Matching

Edição do método **saudar(String nome)**

Remoção do método **saudar(String nome)**
Adição do método **saudar()**

Editar um nó que foi removido é um conflito

```
double saudar(String nome) {
    System.out.println("Hello, " + nome);
}
```

```
double saudar() {
    System.out.println("Olá, humano");
}
```

```
double saudar() {
    System.out.println("Olá, humano");
}

<<<<<<<<
double saudar(String nome) {
    System.out.println("Olá, " + nome);
}
=======
>>>>>>>
```

jDime evita aFPs de renaming, ao custo de aFNs

Last Merge detecta a interferência ao considerar toda a assinatura do construtor.

Olhando para Mergiraf e Spork, o cenário se inverte: Mergiraf possui mais aFPs; Spork, menos aFNs

Situação	Spork	Mergiraf
Falsos positivos adicionados (aFPs)	150	290
Falsos negativos adicionados (aFNs)	107	62

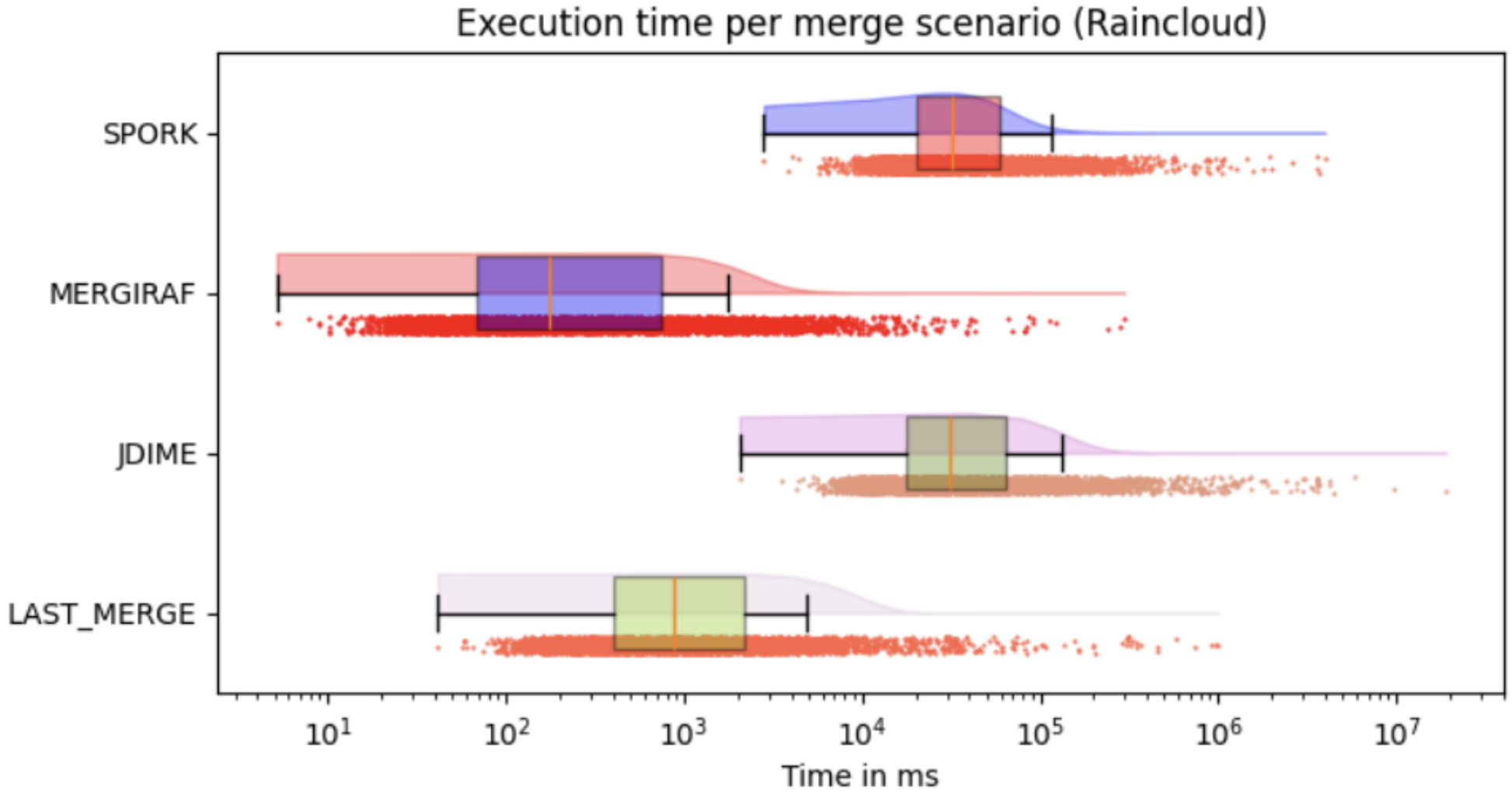
Nos dois casos, as diferenças observadas surgem por diferentes configurações e implementações

- Se as ferramentas tivessem a mesma configuração o resultado observado seria o mesmo;
- As diferenças existentes por implementação não são proibitivas: introduzir auto-tuning em Spork, por exemplo, seria trivial.

Mais importante: as diferenças observadas **não estão relacionadas ao requisito de genericidade**.

Sugere que ferramentas genéricas possuem acurácia semelhante à observada em ferramentas específicas

Além disso, introduzir genericidade não traz consigo um custo computacional proibitivo



Desempenho comparável, apesar das diferenças em design e implementação

- LastMerge e Mergiraf implementadas em Rust; Spork e jDime em Java.
- Mergiraf usa auto-tuning; melhora consideravelmente o desempenho.
- Mesmo considerando uma penalidade de 10x, ferramentas genéricas apresentam desenho comparável ao das específicas
- Uso de memória também não é proibitivo;
  - Cenário com 5500 LoCs, Peaky Memory de Last Merge 35MB vs 825MB de jDime

Ferramentas genéricas podem substituir as específicas, sem prejuízo a acurácia e a performance.

Certo, mas e para outras linguagens de programação?

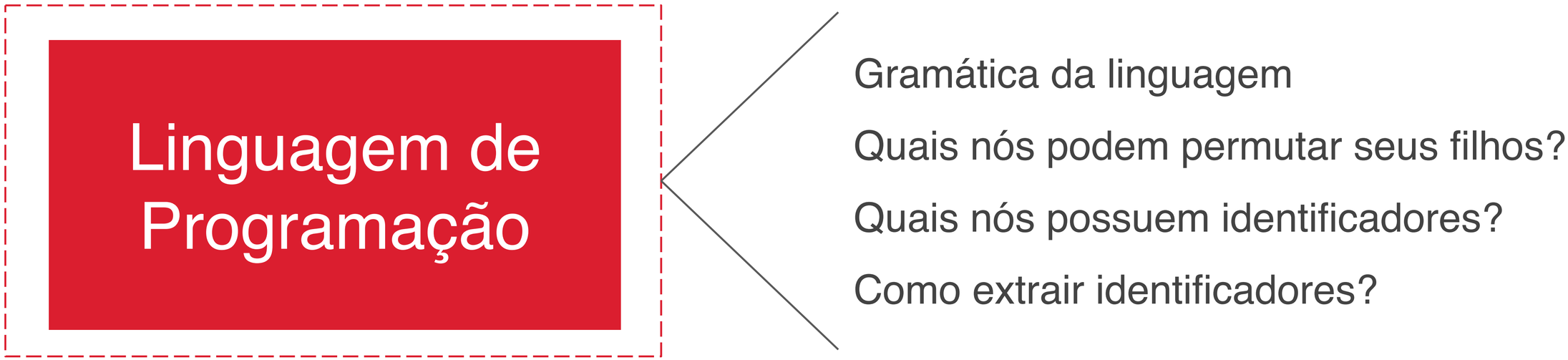
- Nosso estudo focou em comparar ferramentas genéricas instanciadas para Java em comparação com ferramentas específicas;
- **Generalizar é difícil:** ferramentas estruturadas específicas para outras linguagens não estão disponíveis na literatura.

Quão extensível é LastMerge?

- Na prática, a ferramenta precisa ser configurável para uso com outras linguagens de programação;
- Além disso, a configuração deve ser fácil de fazer (plug and play);

Quão difícil é instanciar LastMerge para uso com outras linguagens de programação?

Construindo a interface de configuração: isolar os aspectos relacionados à linguagem de programação



Utilizando casamento de padrão com Tree Sitter Queries

```
class Account {
  // ...
}
```

Parses to

```
(program
  (class_declaration
    name: (identifier)
    body: (class_body)))
```

```
(class_declaration (identifier) @class_name)
```

Permitem encontrar padrões na árvore de maneira agnóstica a linguagem.

Instanciando Last Merge para C#

- Utilização da gramática oficial mantida pelo time do Tree Sitter.
- Identificação de nós que podem permutar filhos (classes, interfaces, etc.)
- Especificar como extrair identificadores para nós (assinaturas de métodos, nomes de classes e propriedades, etc.)

Tree Sitter Playground ajuda a facilitar o entendimento da árvore

```
1  class Account {
2      double currency;
3
4      bool withdrawn(double amount) {
5          // ...
6      }
7  }
```

Tree

0.2 ms

```
compilation_unit [0, 0] - [7, 0]
  class_declaration [0, 0] - [6, 1]
    name: identifier [0, 6] - [0, 13]
    body: declaration_list [0, 14] - [6, 1]
      field_declaration [1, 1] - [1, 17]
        variable_declaration [1, 1] - [1, 16]
          type: predefined_type [1, 1] - [1, 7]
          variable_declarator [1, 8] - [1, 16]
            name: identifier [1, 8] - [1, 16]
      method_declaration [3, 4] - [5, 5]
        returns: predefined_type [3, 4] - [3, 8]
        name: identifier [3, 9] - [3, 18]
        parameters: parameter_list [3, 18] - [3, 33]
          parameter [3, 19] - [3, 32]
            type: predefined_type [3, 19] - [3, 25]
            name: identifier [3, 26] - [3, 32]
        body: block [3, 34] - [5, 5]
          comment [4, 5] - [4, 11]
```

```
1  class Account {
2      private int amount = 0;
3  }
```

Query

```
1  (variable_declarator name: _ @name)
```

Especificação de queries

Identificação de nós

cin.ufpe.br

Percepção: instanciar Last Merge é simples

- Um desenvolvedor que conhece a linguagem deve levar de 2-4 horas;
- Configuração sucinta: PR que adiciona possui apenas 100 linhas;
- Mais importante: configuração pode ser feita de forma incremental
- Existem linguagens em que a instanciação é limitada?

Structured merge

Merge and Code Review

Paulo Borba
Informatics Center
Federal University of Pernambuco

pauloborba.cin.ufpe.br