CAST

Mariam Arutunian
Matevos Mehrabyan
Sevak Sargsyan
Hayk Aslanyan

Precise Code Fragment Clone Detection

VALID 2024
September 29, 2024 to October 03, 2024 - Venice, Italy

About us

Center of Advanced Software Technologies, Armenia

- Members (~50 and growing)
- Research and development, research projects with leading companies
- Publications (40+, Scopus, Web of Science)
  - Program analysis, software security
  - NLP, ECG, medical data analysis
  - Autonomous systems and robotics
- Education

Motivation

Identifying copied code fragments is vital for software

- Management
- Maintenance
- Security

Applications

- Software plagiarism detection
- Malware detection and classification
- Finding known vulnerabilities and avoiding bug propagation

Facts

Studies show that

- About 20% of code is duplicated in software packages [1]
    - Copy-Paste
    - Compiler optimizations like inlining, transformations.
- Over 96% of commercial software packages incorporate open-source code [2]
- 7,800 open-source projects has shown that 44% of them have at least one pair of identical code fragments [3]

Source Code Clones

Original	Type 1	Type 2	Type 3	Type 4

float sum = 0.0;
for (int i = 0; i<n; i++) {
  sum = sum + F[i];
}
	float sum = 0.0; **// Comment**
	for (int i = 0; i<n; i++) {
	____ sum = sum + F[i];
	}
	
	▪ Comments
	▪ Whitespaces
		int sum1 = 0; **// Comment**
		for (int i = 0; i<n; i++) {
		____ sum1 = sum1 + F[i];
		}
		
		▪ Includes Type 1
		▪ Identifiers
		▪ Literals
		▪ Types
			int prod = 1; **// Comment**
			for (int i = 0; i<n; i++) {
			____ prod = prod * F[i];
			}
			
			▪ Includes Type 2
			▪ Instructions addition
			▪ Instructions deletion
			▪ instructions modification
				int factorial_rec (int n) {
				 if (n <= 1) {
				  return 1;
				 } else {
				  return n * factorial_rec (n - 1);
				 }
				}
				
				▪ The same calculation, but uses different instructions

Binary Code Clones

Original	BinType 1	BinType 2	BinType 3	BinType 4

mov [ebp+var_1], 5
mov eax, [ebp+var_1]
iadd eax, [ebp+var_4]
	mov [ebp+var_1], 5
	mov eax, [ebp+var_1]
	iadd eax, [ebp+var_4]
	
	- Identical
		mov [ebp+var_1], 10
		mov ecx, [ebp+var_1]
		iadd ecx, [ebp+var_4]
		
		- Includes Type 1
		- Registers
		- Literals
		- Operand size
			~~mov [ebp+var_1], 10~~
			mov ecx, [ebp+var_1]
			iadd ecx, [ebp+var_4]
			
			- Includes Type 2
			- Instructions addition
			- Instructions deletion
			- instructions modification
				factorial_O3:
				movl $1, %eax
				cmpl $1, %edi
				jle .L1
				.p2align 4,,10
				.p2align 3
				.L2:
				movl %edi, %edx
				subl $1, %edi
				imull %edx, %eax
				cmpl $1, %edi
				jne .L2
				.L1:
				ret
				- The same calculation, but uses different instructions

Problem Description

Despite the variety of code clone detection methods and tools:

01 Only few can detect clones of fragments rather than whole functions

02 There is no unified approach: either source or binary code clone detection

Code Clone Detection Techniques

Text-based

- Two code fragments are compared in the form of text/strings
- Finds Type 1 clones

Token-based

- The entire code is transformed into a sequence of tokens
- More robust against code changes than the text-based techniques
- Finds Type 1 and Type 2 clones

Metrics-based

- Different types of metrics are calculated for code fragments usually on some graph representation, such as AST or PDG.
- Suffers in precision and produces many false positives

Tree-based

- Uses parse trees or AST of the analyzable code
- Tree matching algorithm for similar subtree detection
- Finds Type 1, Type 2 and Type 3 clones
- Low precision for Type 3 clones detection

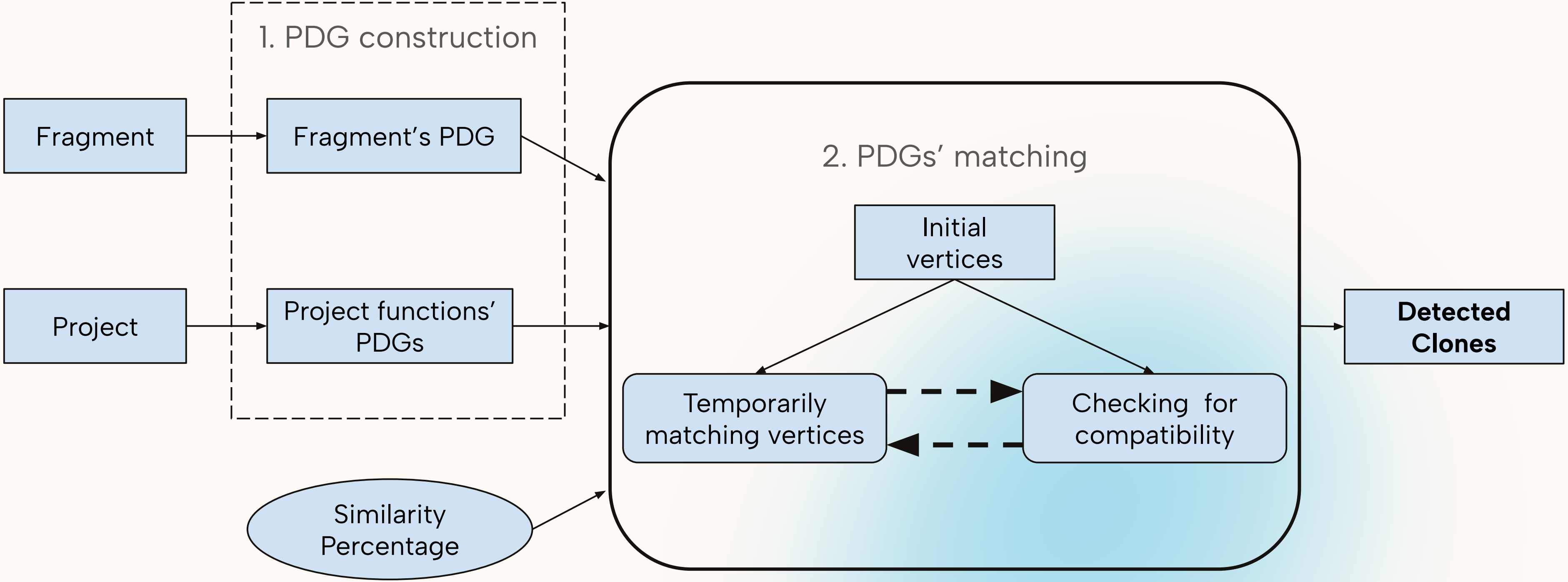
Graph-based

- Maximal isomorphic or similar subgraphs are searched on PDGs or CFGs
- Are robust to the insertion and deletion of code, reordered instructions, intertwined and non-contiguous code.

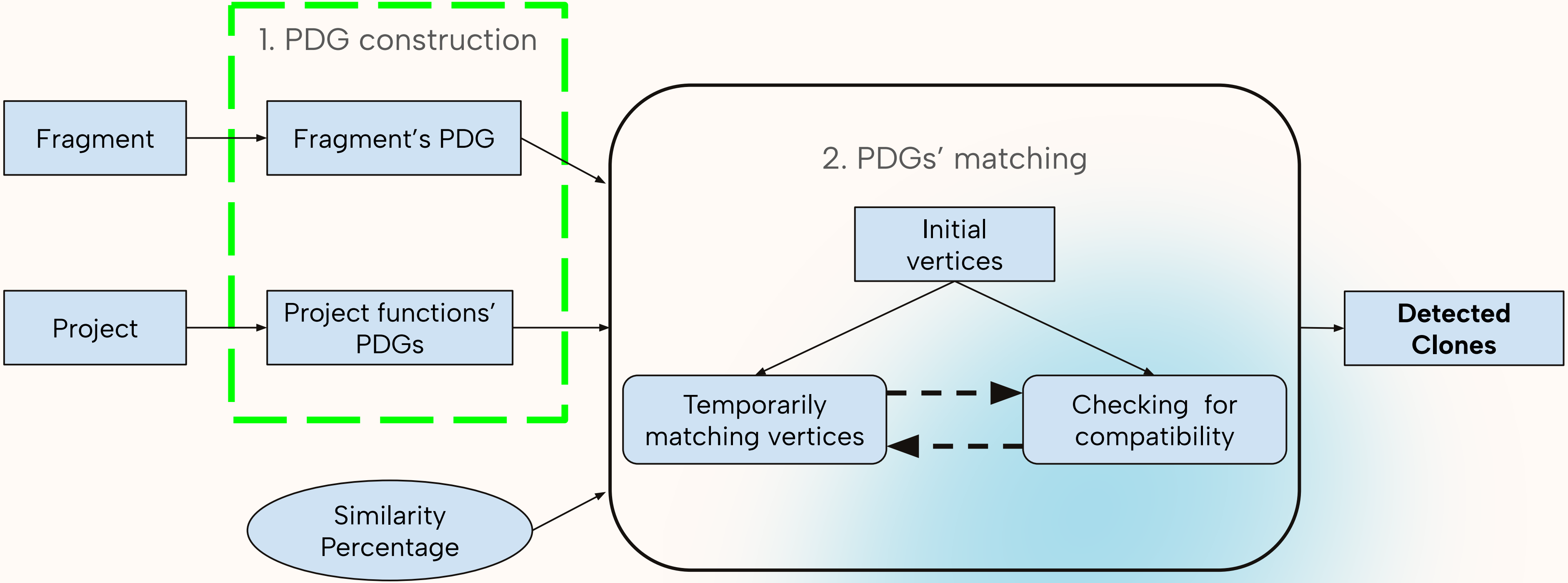
Machine learning-based

- The focus is on training models to classify or cluster similar code fragments
- Needs a large dataset containing similar and dissimilar examples of codes
- Finds Type 1 , Type 2, Type 3, Type 4 clones,

Architecture of The Method



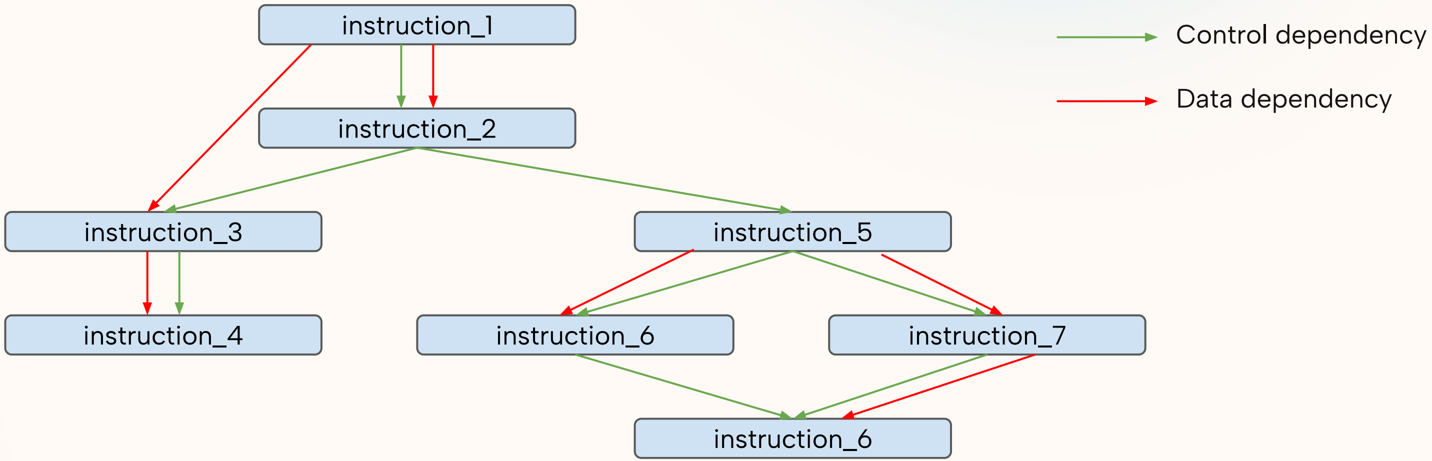
Architecture of The Method



Program Dependency Graph

Program Dependency Graph (PDG) is a directed graph where

- Vertices are instructions of Intermediate Representation (IR)
- Edges are data and control dependencies between instructions

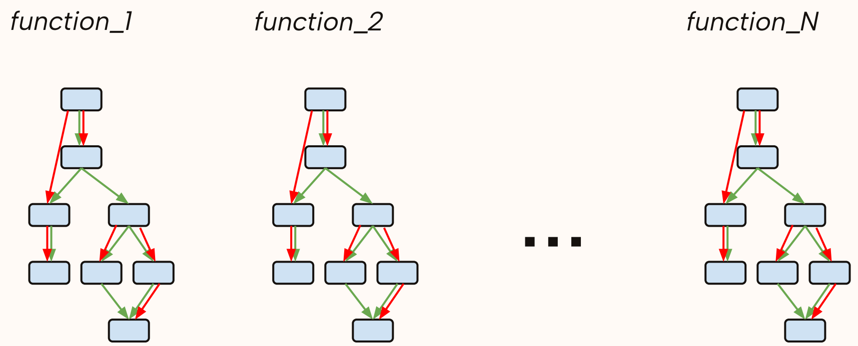


PDG construction

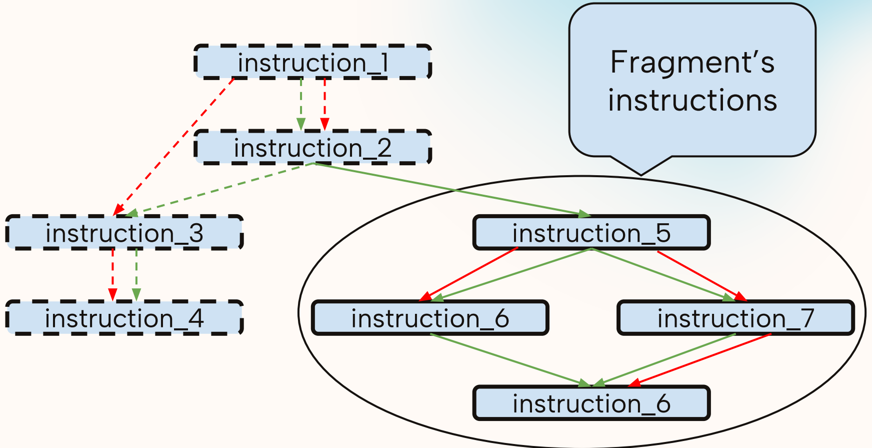
PDGs are constructed

- For all functions of the project to analyze
- For the code fragment

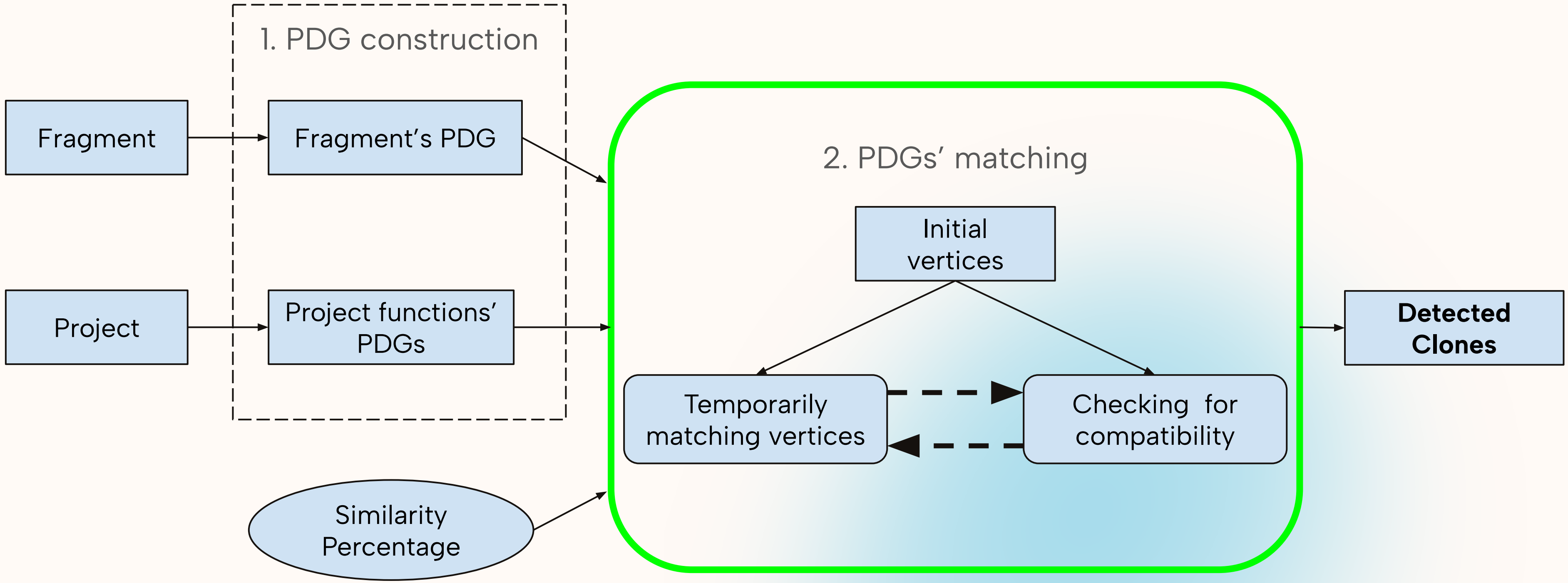
Project's PDGs



Fragment's PDGs



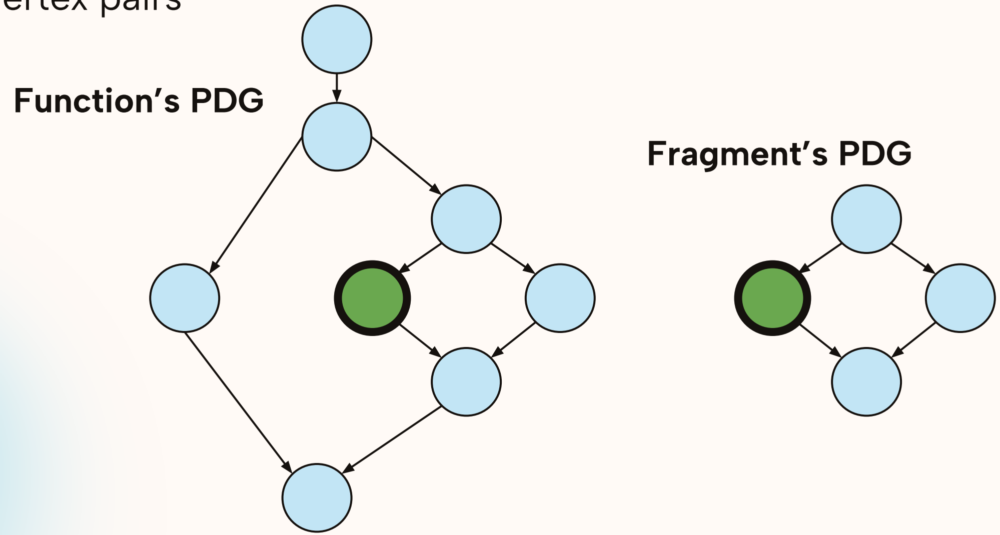
Architecture of The Method



Graphs’ Matching

Matching algorithm has two main phases:

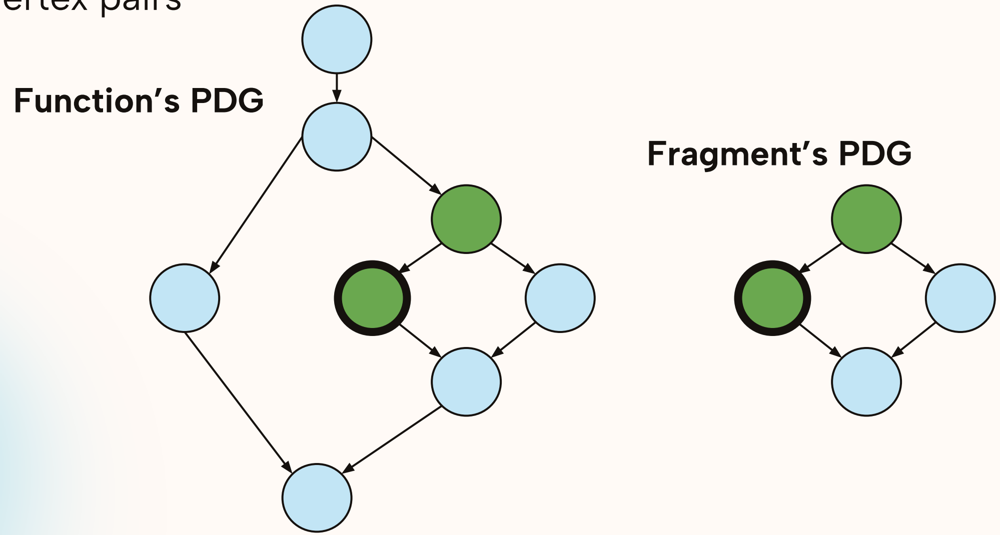
- Construction of the set of initial matched vertex pairs
- Iterative expansion of matched vertex pairs



Graphs' Matching

Matching algorithm has two main phases:

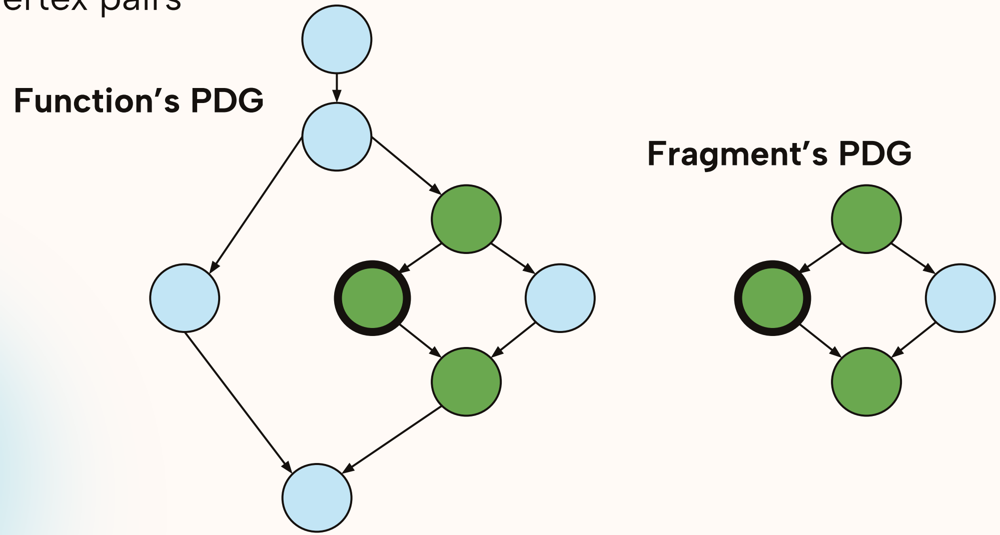
- Construction of the set of initial matched vertex pairs
- Iterative expansion of matched vertex pairs



Graphs' Matching

Matching algorithm has two main phases:

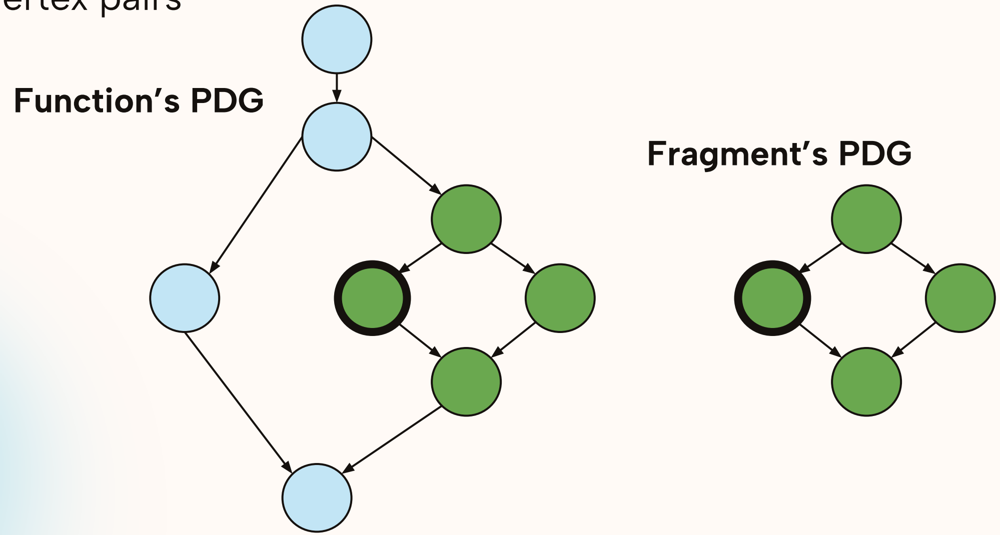
- Construction of the set of initial matched vertex pairs
- Iterative expansion of matched vertex pairs



Graphs' Matching

Matching algorithm has two main phases:

- Construction of the set of initial matched vertex pairs
- Iterative expansion of matched vertex pairs

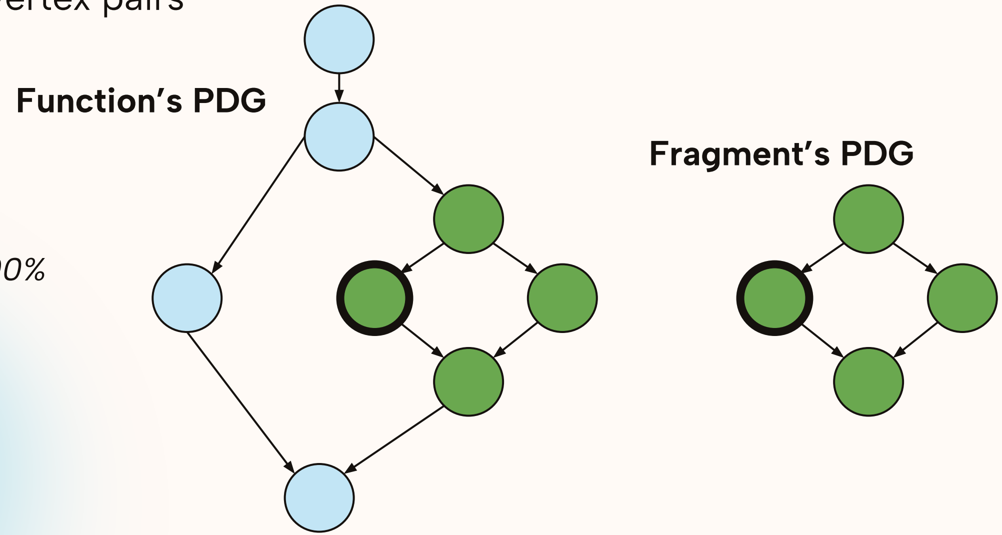


Graphs' Matching

Matching algorithm has two main phases:

- Construction of the set of initial matched vertex pairs
- Iterative expansion of matched vertex pairs

$$similarity = \frac{matched\ common\ vertices\ count}{fragment\ PDG's\ vertices\ count} * 100\%$$



Graphs' Matching - Initial Vertices Selection

Based on experimental evaluation, the following subroutines were chosen for initial vertices pair selection:

01 All vertices (v, v^*) with no incoming edges in both PDGs, where $v \in fragment\_PDG$, $v^* \in function\_PDG$

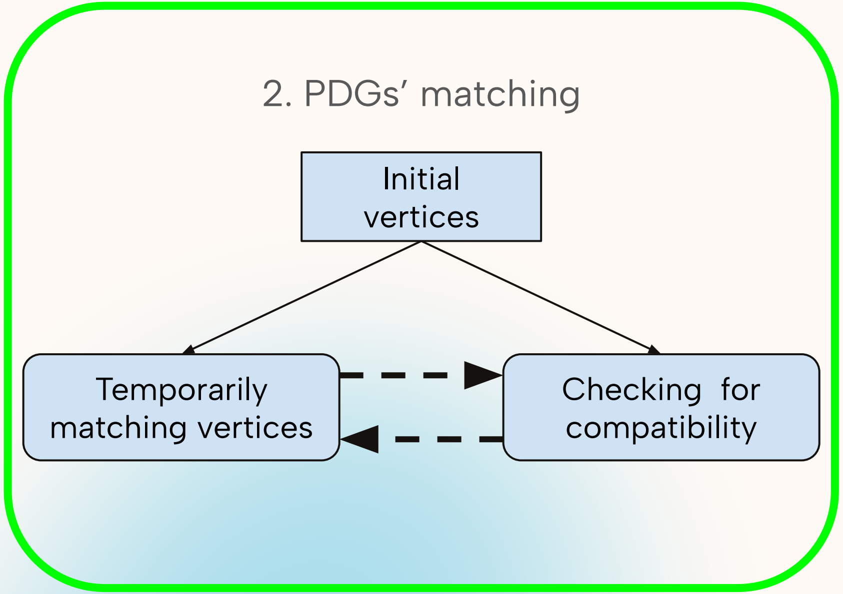
02 All vertices (v, v^*), where $v \in fragment\_PDG$ and $|pred\_ctrl(v)|$ is the maximum. $v^* \in function\_PDG$ and $|pred\_ctrl(v^*)| \geq |pred\_ctrl(v)|$

03 All vertices (v, v^*), where $v \in fragment\_PDG$ and $pred\_data(v)$ is the maximum. $v^* \in function\_PDG$ and $pred\_data(v^*) \geq pred\_data(v)$

Graphs' Matching

1. **Temporarily matching vertices**
   - Five subroutines.
2. **Checking for compatibility**
   - The temporarily matched pairs are checked against two conditions and some of them may be filtered out.

The matching process is complete when no new pairs of vertices are temporarily matched



Temporarily Matching Subroutines

- Based on incoming and outgoing control flow
- Based on basic block
- Based on predecessor and successor basic blocks
- Based on incoming and outgoing data flow
- Based on initial_pairs

Temporarily Matching Subroutines

Temporarily matching is allow for two vertices (u, u*):

01 opcode(u) == opcode(u*)

02 |pred_ctrl(u)| == |pred_ctrl(u*)|

03 |succ_ctrl(u)| == |succ_ctrl(u*)|

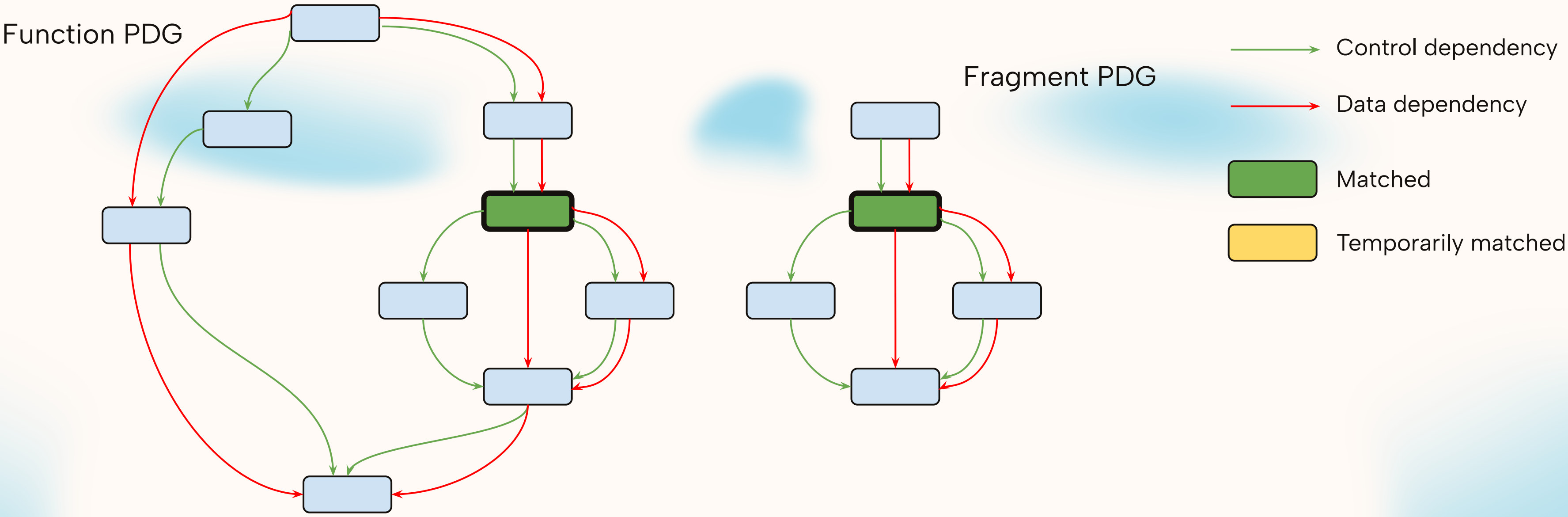
04 (u, u*) ∉ matched_pairs

05 (u, u*) ∉ incompatible_pairs

#1 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

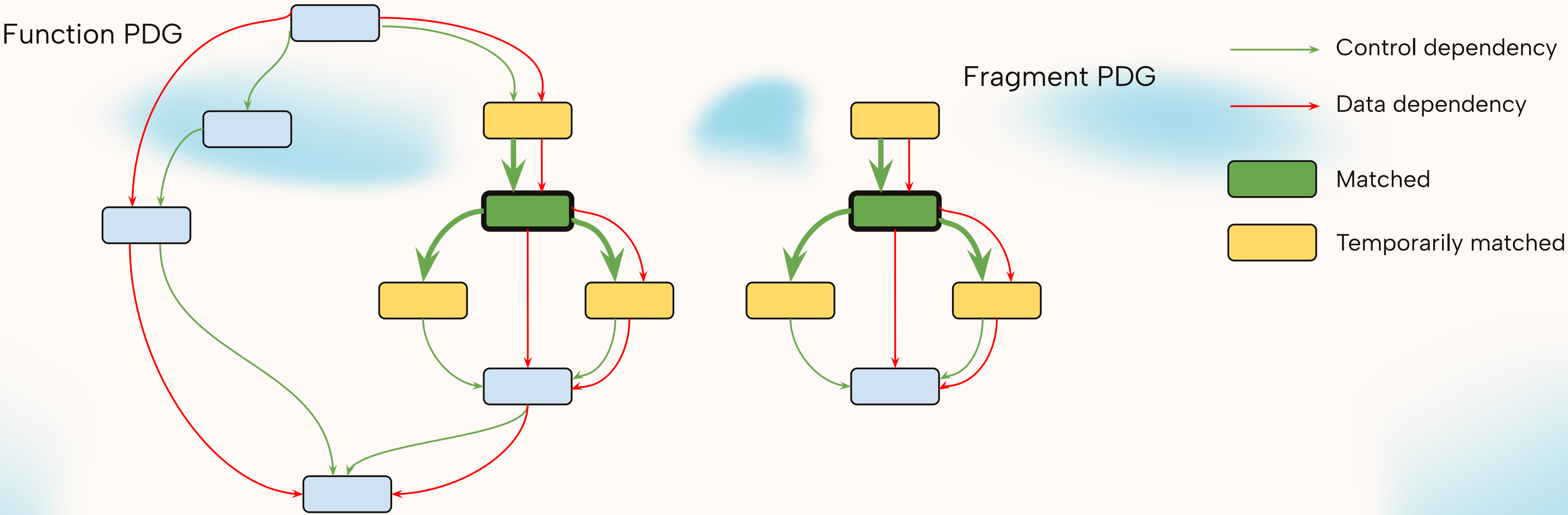
- *(u, u*), where u ∈ <u>pred_ctrl(v)</u>, u* ∈ <u>pred_ctrl(v*)</u>, TMP_MATCH_ALLOWED((u, u*)) == true*
- *(s, s*), where s ∈ <u>succ_ctrl(v)</u>, s* ∈ <u>succ_ctrl(v*)</u>, TMP_MATCH_ALLOWED((s, s*)) == true*



#1 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

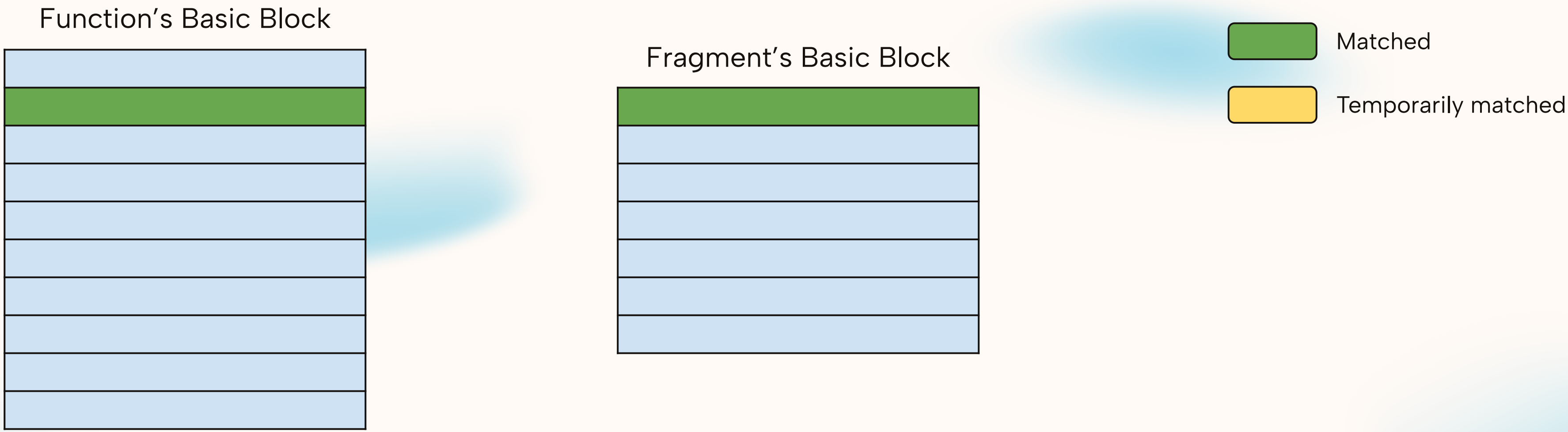
- *(u, u*), where u ∈ pred_ctrl(v), u* ∈ pred_ctrl(v*), TMP_MATCH_ALLOWED((u, u*)) == true*
- *(s, s*), where s ∈ succ_ctrl(v), s* ∈ succ_ctrl(v*), TMP_MATCH_ALLOWED((s, s*)) == true*



#2 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

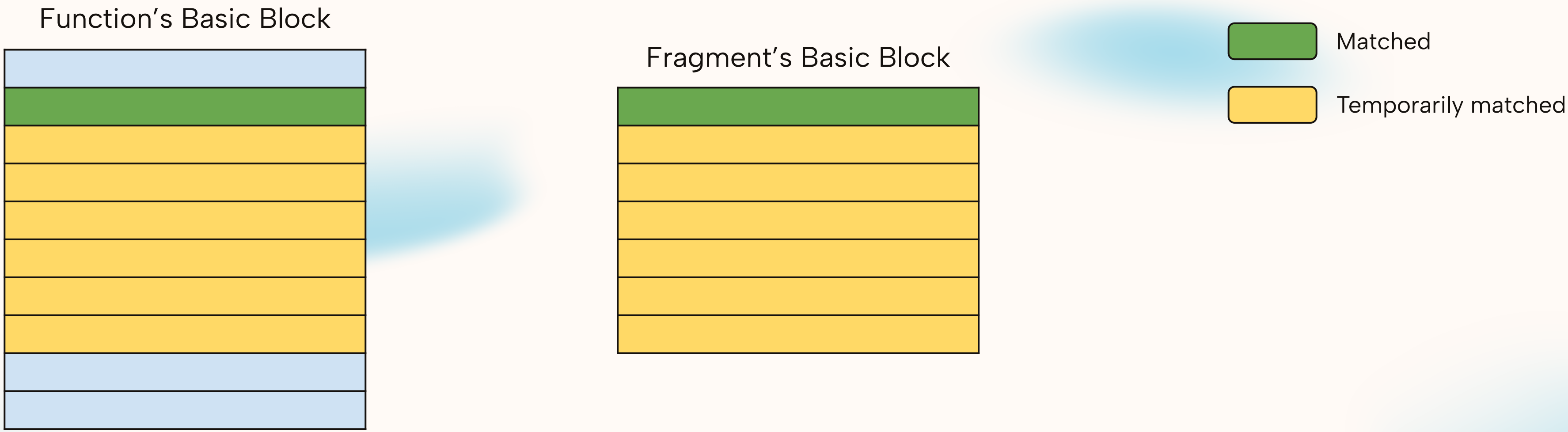
- *(u, u*), where u ∈ <u>bb(v)</u>, u* ∈ <u>bb(v*)</u>, TMP_MATCH_ALLOWED((u, u*)) == true*



#2 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

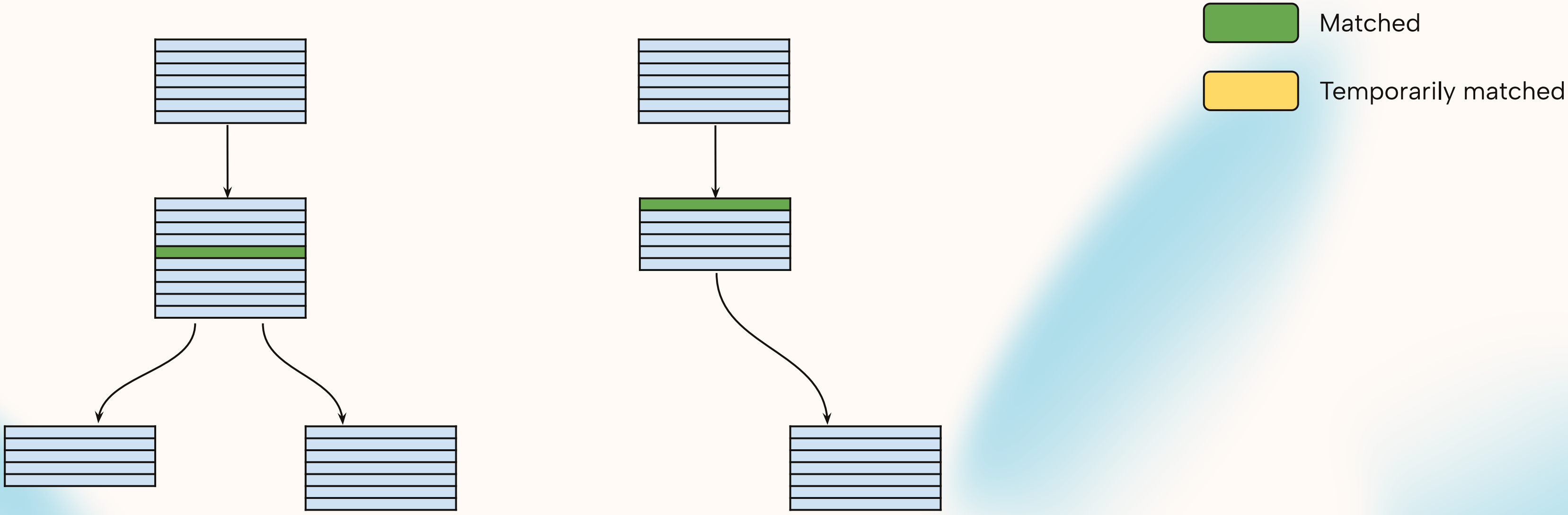
- *(u, u*), where u ∈ bb(v), u* ∈ bb(v*), TMP_MATCH_ALLOWED((u, u*)) == true*



#3 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

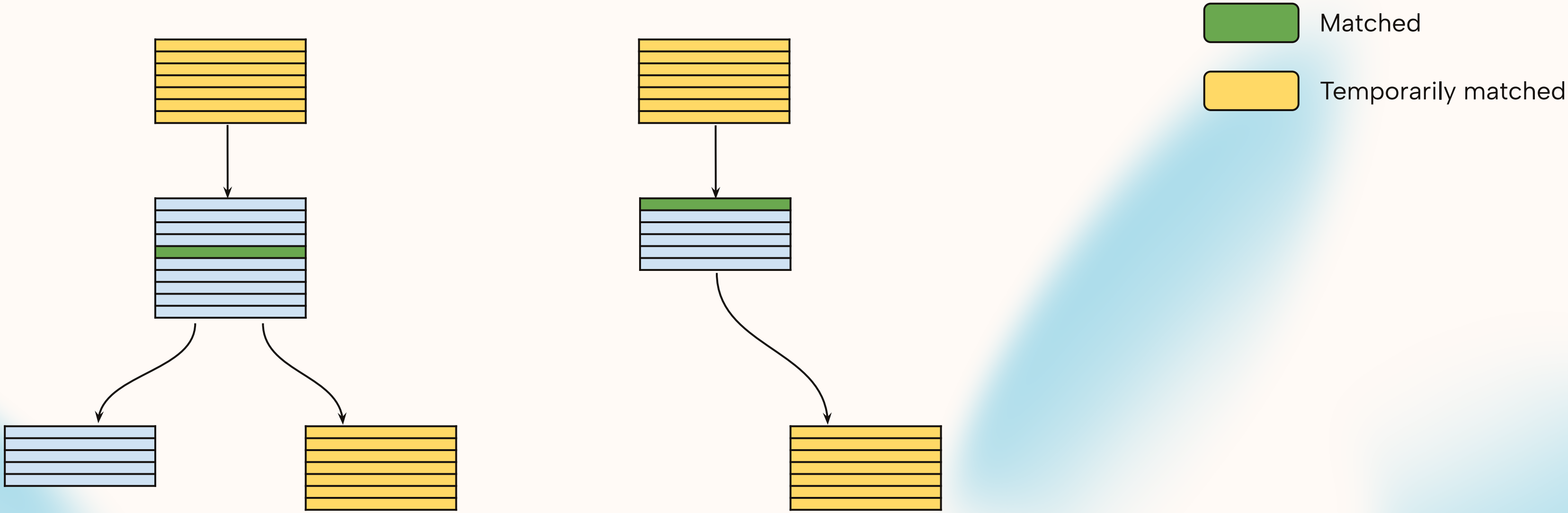
- *(u, u*), where u ∈ pred_bb(v), u* ∈ pred_bb(v*), TMP_MATCH_ALLOWED((u, u*)) == true*
- *(s, s*), where s ∈ succ_bb(v), s* ∈ succ_bb(v*), TMP_MATCH_ALLOWED((s, s*)) == true*



#3 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

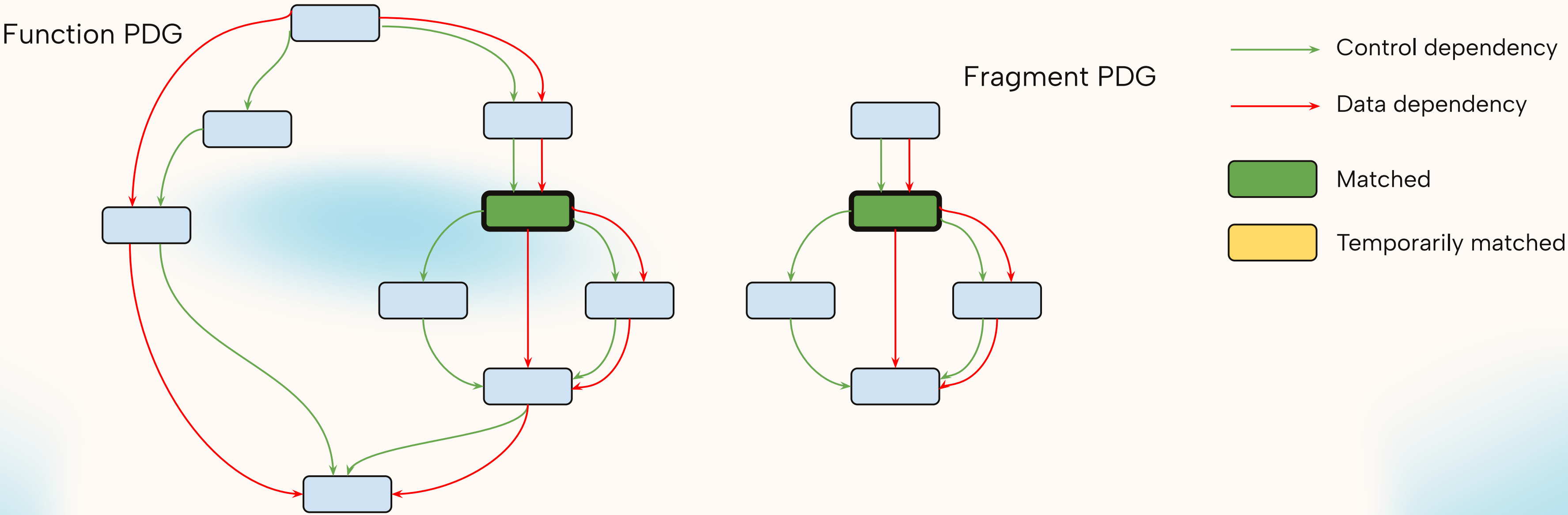
- *(u, u*), where u ∈ <u>pred_bb</u>(v), u* ∈ <u>pred_bb</u>(v*), TMP_MATCH_ALLOWED((u, u*)) == true*
- *(s, s*), where s ∈ <u>succ_bb</u>(v), s* ∈ <u>succ_bb</u>(v*), TMP_MATCH_ALLOWED((s, s*)) == true*



#4 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

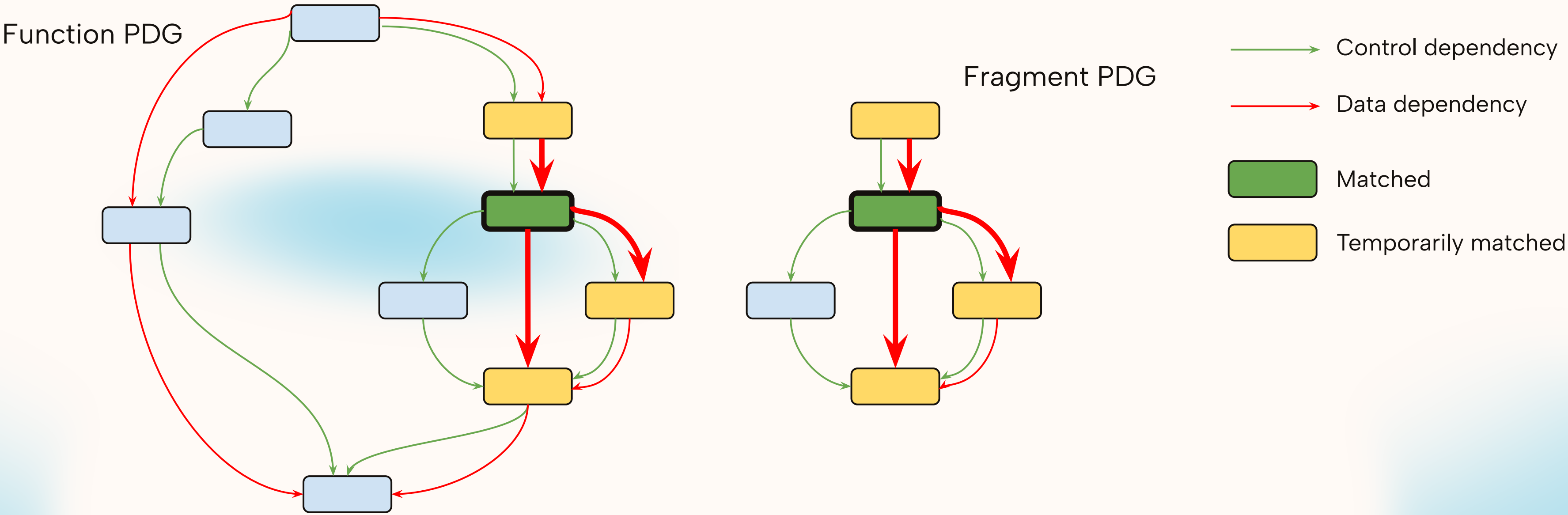
- *(u, u*), where u ∈ <u>pred_data</u>(v), u* ∈ <u>pred_data</u>(v*), TMP_MATCH_ALLOWED((u, u*)) == true*
- *(s, s*), where s ∈ <u>succ_data</u>(v), s* ∈ <u>succ_data</u>(v*), TMP_MATCH_ALLOWED((s, s*)) == true*



#4 Temporarily Matching Subroutine

For each pair (v, v*) ∈ matched_pairs, temporarily match vertices:

- *(u, u*), where u ∈ pred_data(v), u* ∈ pred_data(v*), TMP_MATCH_ALLOWED((u, u*)) == true*
- *(s, s*), where s ∈ succ_data(v), s* ∈ succ_data(v*), TMP_MATCH_ALLOWED((s, s*)) == true*



#5 Temporarily Matching Subroutine

Temporarily match vertices *(u, u*)* ∈ *initial_pairs*, *(u, u*)* ∉ *matched_pairs*, *(u, u*)* ∉ *incompatible_pairs*

Function's initial vertices : **[v8, v14, v98]**

Fragment's initial vertices : **[u4, u72]**

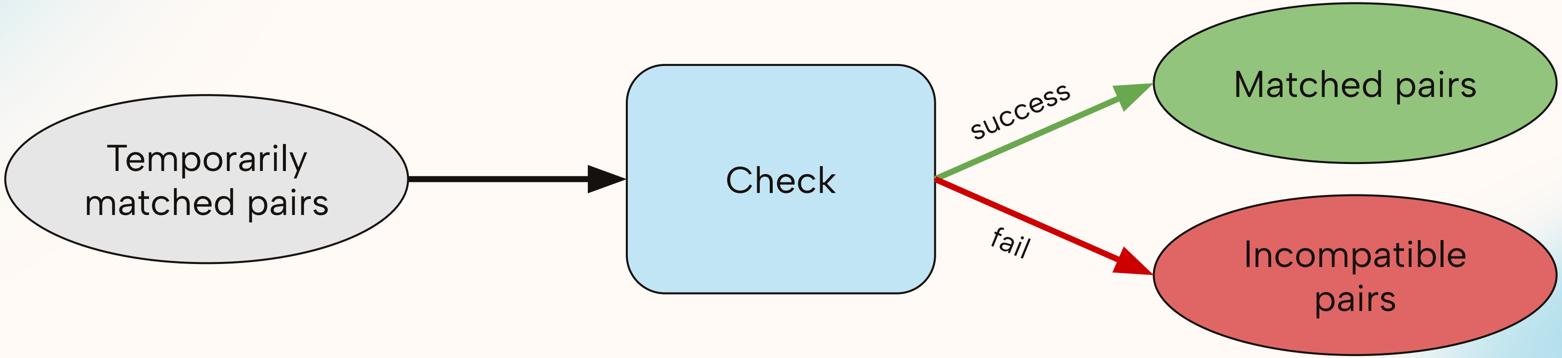
#5 Temporarily Matching Subroutine

Temporarily match vertices *(u, u*)* ∈ *initial_pairs*, *(u, u*)* ∉ *matched_pairs*, *(u, u*)* ∉ *incompatible_pairs*

Function's initial vertices : **[v8, v14, v98]**

Fragment's initial vertices : **[u4, u72]**

Checking for Compatibility



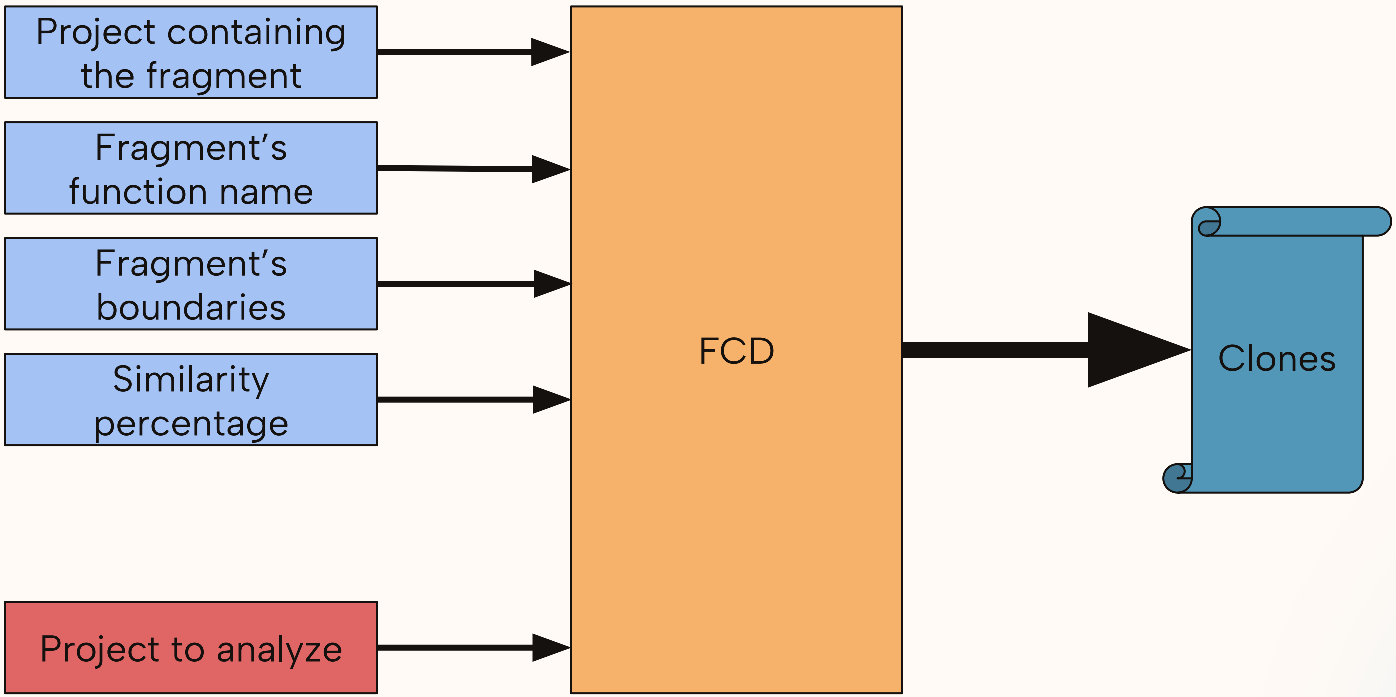
01 pred_condition(v, v*) **fails** if:

$\exists p \in$ pred_ctrl(v), (p, p*) $\in$ matched_pairs, $\nexists p^* \in$ pred_ctrl(v*),

02 succ_condition(v, v*) **fails** if:

$\exists s \in$ succ_ctrl(v), (s, s*) $\in$ matched_pairs, $\nexists s^* \in$ succ_ctrl(v*),

Implementation



Used Intermediate Representations

- Source Code – LLVM intermediate representation
- Binary Code – REIL intermediate representation

Testing System

The testing system creates PDGs of real-world projects, duplicates each PDG, removes some vertices from it and considers the original one as a fragment.

- It randomly selects a basic block and removes it vertices until the desired percentage is reached
- If the desired percentage wasn't reached by removing all vertices of the basic block, another random basic block is selected for vertices removal.
- Predecessor vertices of the removed vertices are connected to their successor vertices.
- Testing was done for 100%, 90%, 80%, and 70% similarity clones.

Source Code Clones Evaluation

Project	C/C++ code lines	Precision	Recall	RMSE	FCD speed
c-ares 1.15.0	61087	97.5	95.2	6.1	29s
jasper 1.900.1	28279	95.4	93	6	15s
openssl 1.0.2t	310922	97	95.1	7.7	2s
rsync 3.1.3	44832	96	91.9	10.7	26s

Binary Code Clones Evaluation (1)

Project	Size of the binary	Architecture	Precision	Recall	RMSE	FCD speed
libcares 2.3.0 (c-ares 1.15.0)	86 KiB	x86–64	98.9	95.6	4.6	41s
libcares 2.3.0 (c-ares 1.15.0)	96 KiB	x86	97.9	93.4	5.5	43s
libcares 2.3.0 (c-ares 1.15.0)	146 KiB	ARM	98.9	95.6	4.6	49s
jasper 1.900.1	1.5 MiB	x86–64	96	92.1	5.4	3m 5s
jasper 1.900.1	368 KiB	x86	95	90	6.5	2m 1s
jasper 1.900.1	478 KiB	ARM	94.1	89.8	6.1	2m 8s

Binary Code Clones Evaluation (2)

Project	Size of the binary	Architecture	Precision	Recall	RMSE	FCD speed
openssl 1.0.2t	536 KiB	x86-64	99.9	98.1	3.8	1m 10s
openssl 1.0.2t	507 KiB	x86	98.8	95.8	3.9	0m 57s
openssl 1.0.2t	634 KiB	ARM	97.9	95.6	4.4	1m 25s
rsync 1.3.2	1.7 MiB	x86-64	96	91	6.6	3m 34s
rsync 1.3.2	1.6 MiB	x86	94.9	88.9	6.7	3m 21s
rsync 1.3.2	1.8 MiB	ARM	94.1	88.8	7.4	3m 58s

Detected Clones of Existing CVEs

Found 14 bugs

- 7 of them are already accepted
- 2 of them are rejected as the maintainers use the projects as tests

Openly accessible discoveries

- CMake – https://gitlab.kitware.com/cmake/cmake/-/issues/26112
- OpenJPEG https://github.com/uclouvain/openjpeg/issues/1539
- PointCloudLibrary https://github.com/PointCloudLibrary/pcl/issues/6080
- 0ad https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=1036970
- ITK https://github.com/InsightSoftwareConsortium/ITK/issues/4777

Due to security concerns, 2 of our findings remain confidential.

CAST

Do you have any questions?

hayk.aslanyan@rau.am
https://castech.am/

Thanks!