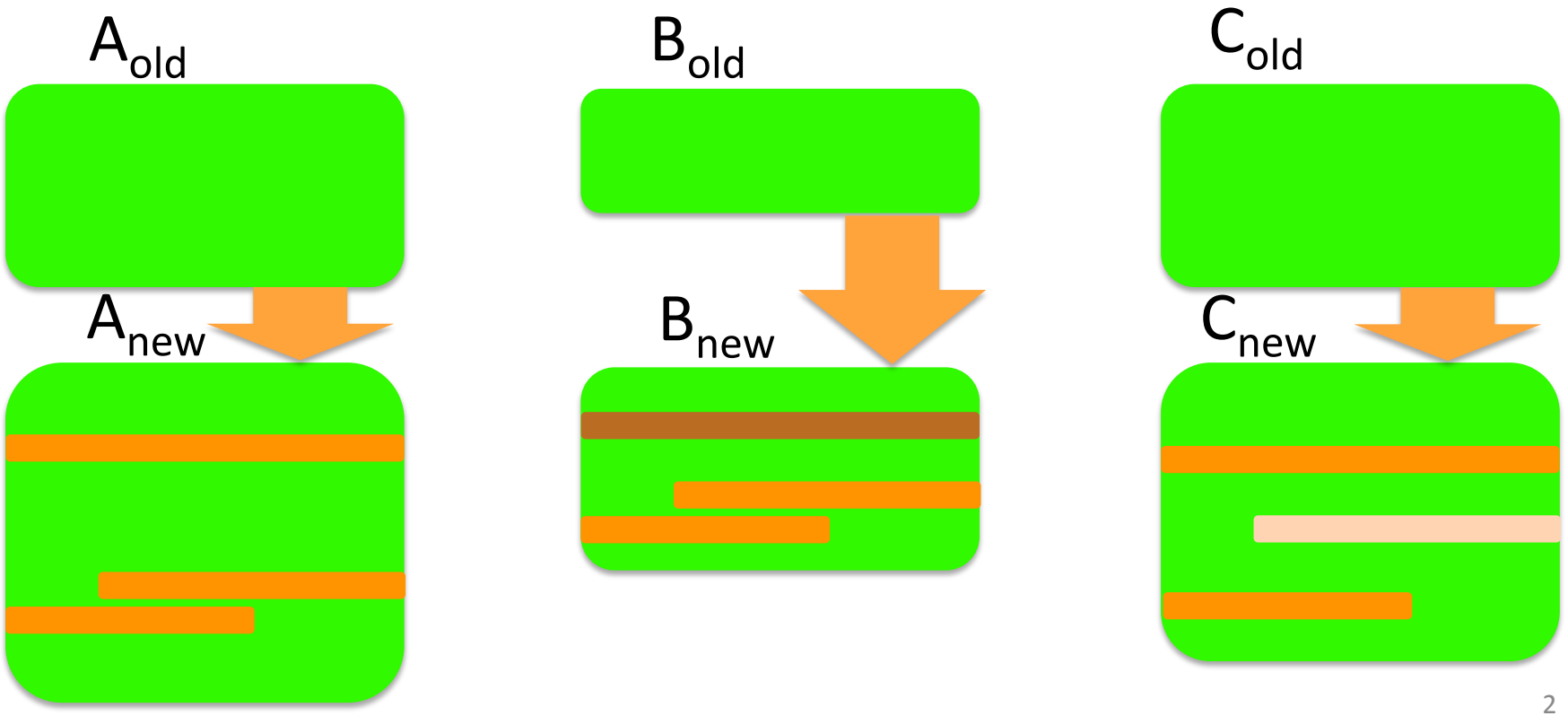
Systematic Editing: Generating Program Transformations from an Example

Na Meng Miryung Kim Kathryn S. McKinley
The University of Texas at Austin

Motivating Scenario

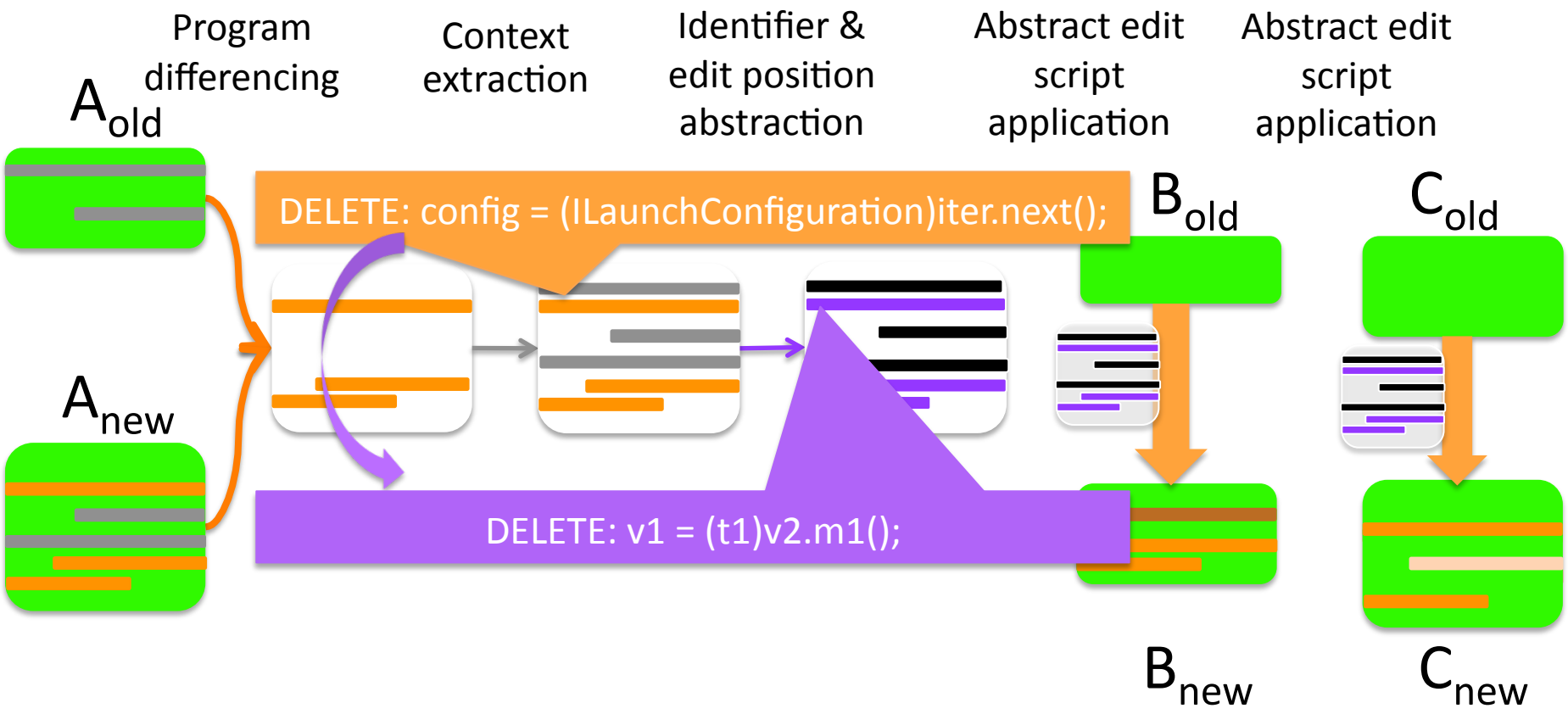
Pat needs to update database transaction code to prevent SQL injection attacks



Systematic Editing

- ***Similar but not identical changes*** to multiple contexts
- Manual, tedious, and error-prone
- Refactoring engines automate pre-defined semantic preserving edits
- Source transformation tools require describing edits in a formal language

Our Solution: Sydit



Sydit improves programmer productivity.

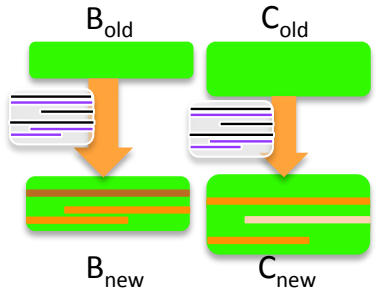
Outline

- Program Transformation Generation Approach
- Evaluation on Open Source Projects
- Related Work
- Conclusion

Approach Overview

- Phase I: Creating Abstract Edit Scripts
  - Step 1. Syntactic Program Differencing
  - Step 2. Edit Context Extraction
  - Step 3. Identifier and Edit Position Abstraction

- Phase II: Applying Abstract Edit Scripts



Step 1. Syntactic Program Differencing

ChangeDistiller(A_{old}, A_{new}) → AST edit script

insert*(u, v, k)*: insert node *u* and position it as the (k+1)th child of node *v*

delete*(u)*: delete node *u*

update*(u, v)*: replace *u* with *v*

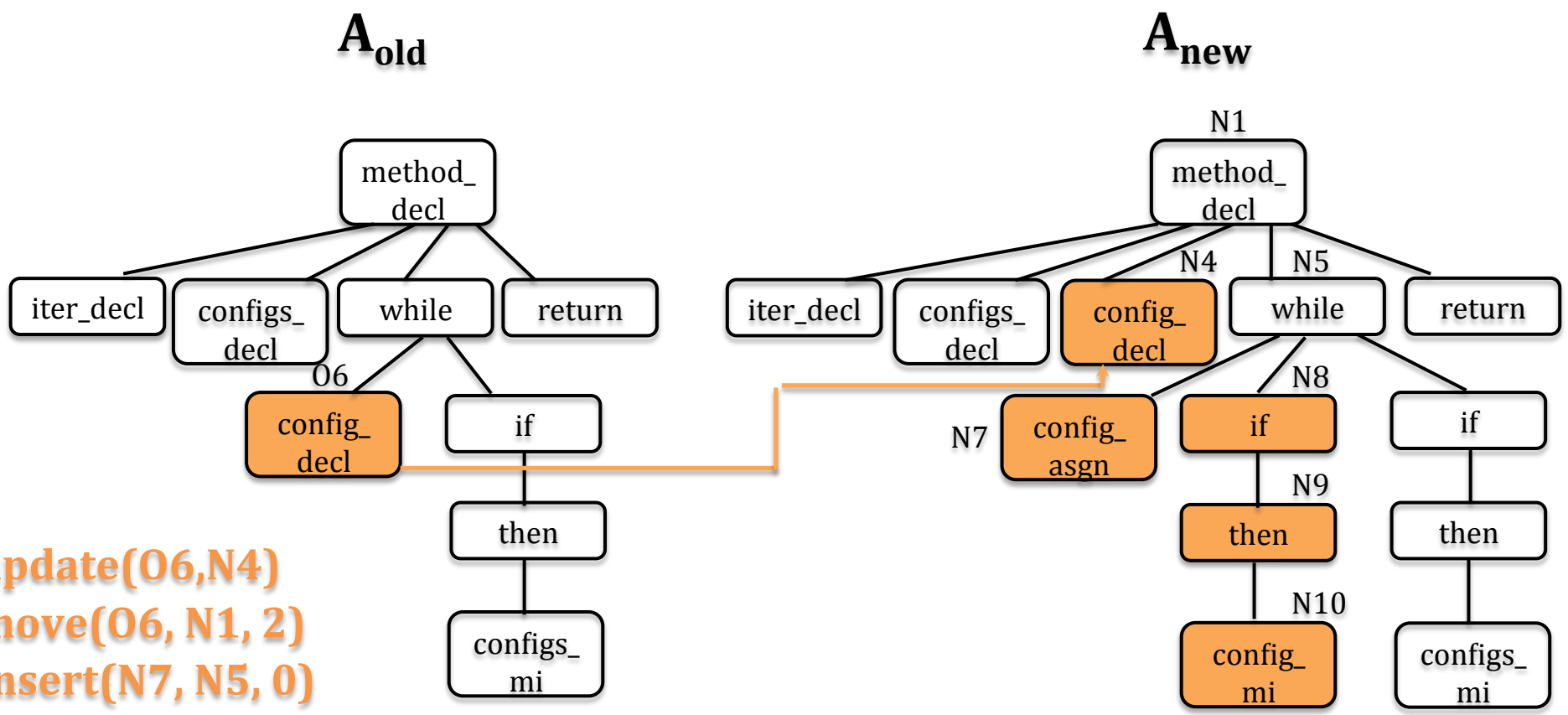
move*(u, v, k)*: delete *u* from its current position and insert *u* as the (k+1)th child of *v*

Example based on *eclipse.debug.core*

A_{old} to A_{new}

```
1.   Iterator iter = getAllLaunchConfigurations().iterator();
2.   List configs = new ArrayList();
3. + ILaunchConfiguration config = null;
4.   while(iter.hasNext()){
5.   - ILaunchConfiguration config = (ILaunchConfiguration)iter.next()
6.   + config = (ILaunchConfiguration)iter.next();
7.   + if(!config.inValid()){
8.   +   config.reset();
9.   + }
10.    if(config.getType.equals(type)){
11.      configs.add(config);
12.    }
13.  }
14.  return (ILaunchConfiguration[])configs.toArray(new
   ILaunchConfiguration[configs.size()]);
```

Step 1. Syntactic Program Differencing



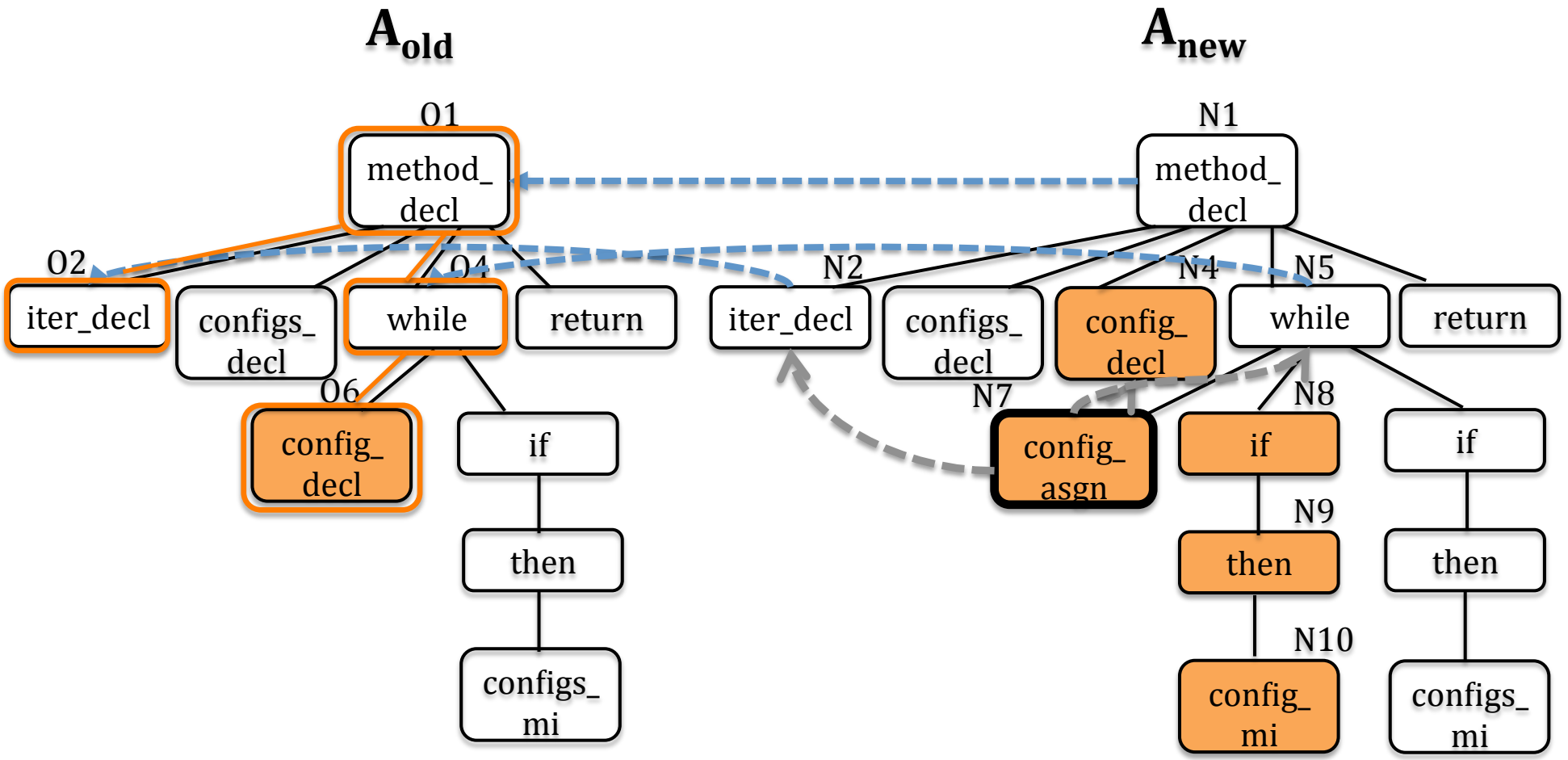
update(O6,N4)
move(O6, N1, 2)
insert(N7, N5, 0)
insert(N8, N5, 1)
insert(N9, N8, 0)
insert(N10, N9, 0)

Step 2. Edit Context Extraction

Identify unchanged nodes on which the edited nodes depend or nodes that depend on the edits

- Containment dependence
- Control dependence
- Data dependence

Step 2. Edit Context Extraction



Step 3. Identifier and Edit Position Abstraction

- Identifier Abstraction

  config = (ILaunchConfiguration) iter.next()

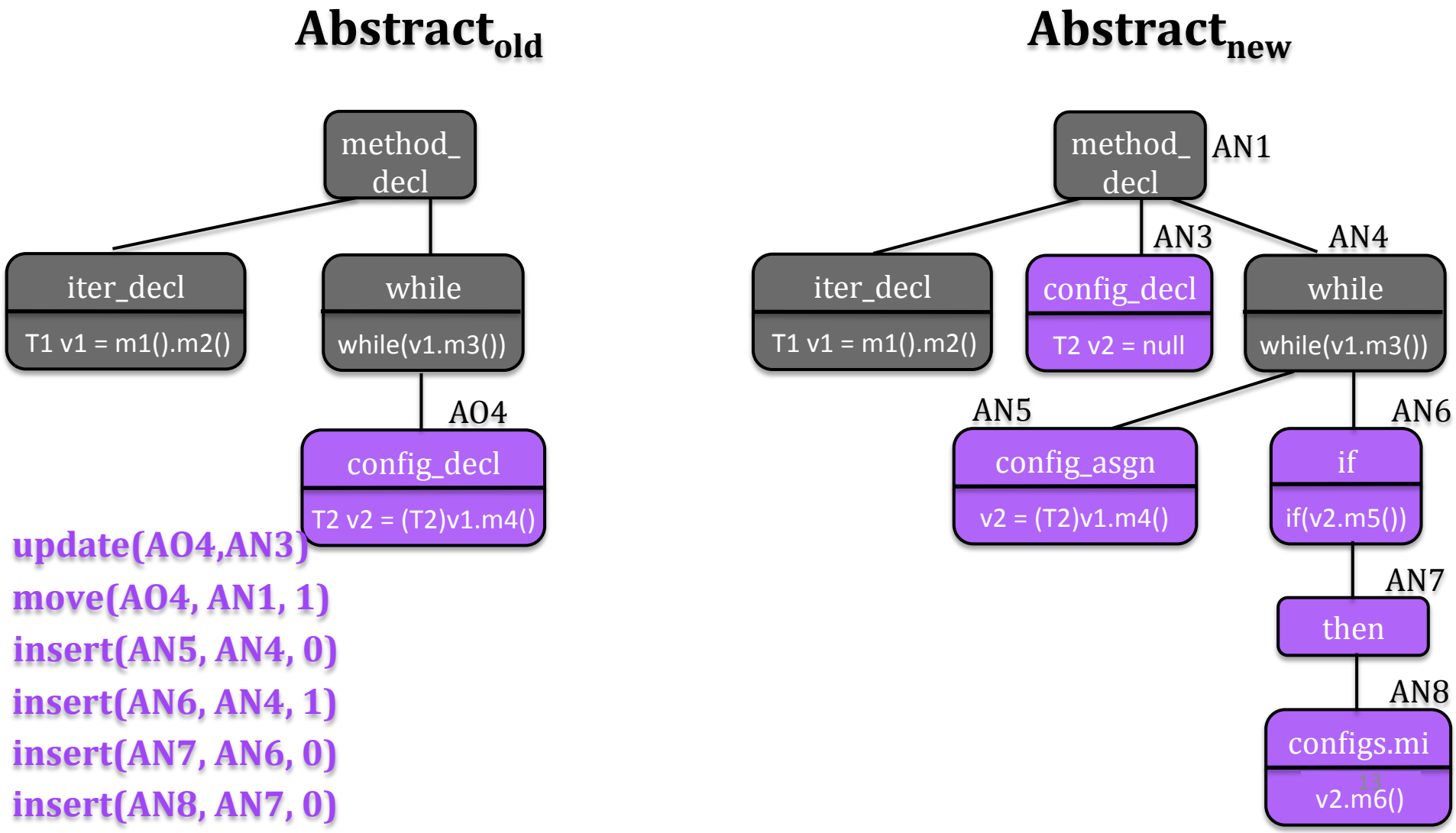
  => v1 = (T1) v2.m1()

- Edit Position Abstraction

  insert (“i++”, “while(i<j)”, 2)

  => insert (“v1++”, “while(v1<v2)”, 1)

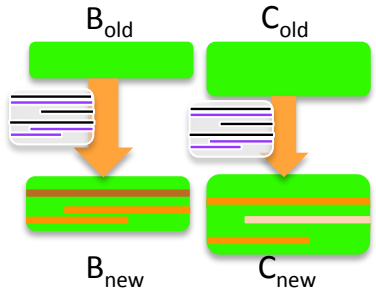
Step 3. Identifier and Edit Position Abstraction



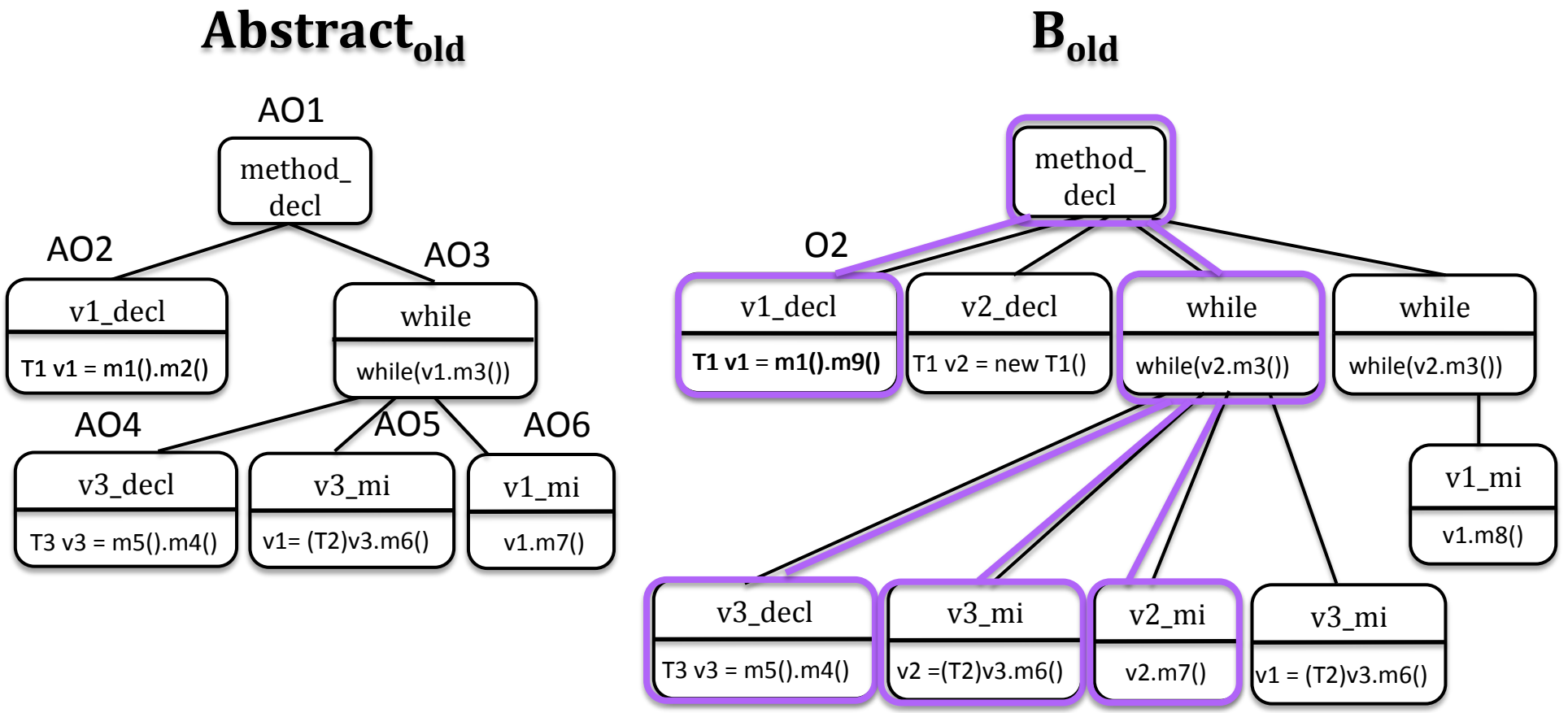
Approach Overview

- Phase I: Creating Abstract Edit Scripts
  - Step 1. Syntactic Program Differencing
  - Step 2. Edit Context Extraction
  - Step 3. Identifier and Edit Position Abstraction

- Phase II: Applying Abstract Edit Scripts
  - Step 4. Context Matching
  - Step 5. Identifier and Edit Position Concretization
  - Step 6. Concrete Edit Application



Step 4. Context Matching

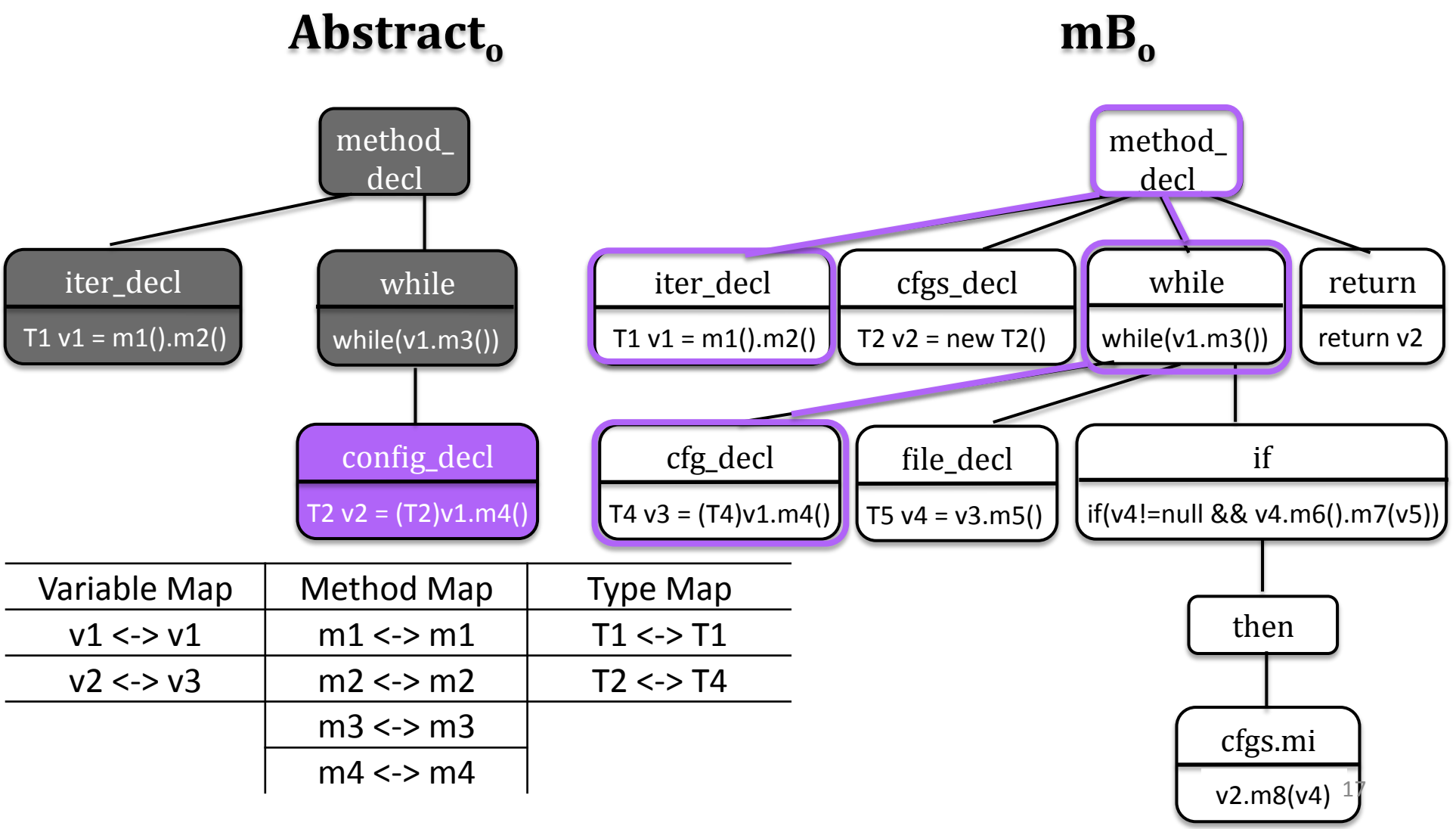


Example based on *eclipse.debug.core*

B_{old}

```
1. Iterator iter = getAllLaunchConfigurations().iterator();
2. List cfgs = new ArrayList();
3. while(iter.hasNext()){
4.   ILaunchConfiguration cfg = (ILaunchConfiguration)iter.next();
5.   IFile file = cfg.getFile();
6.   if(file != null && file.getProject().equals(project)){
7.     cfgs.add(cfg);
8.   }
9. }
10.return cfgs;
```

Step 4. Context Matching



Variable Map	Method Map	Type Map
v1 <-> v1	m1 <-> m1	T1 <-> T1
v2 <-> v3	m2 <-> m2	T2 <-> T4
	m3 <-> m3	
	m4 <-> m4	

Step 5. Identifier and Edit Position Concretization

- Identifier Concretization

  v1 = (T1) v2.m1()

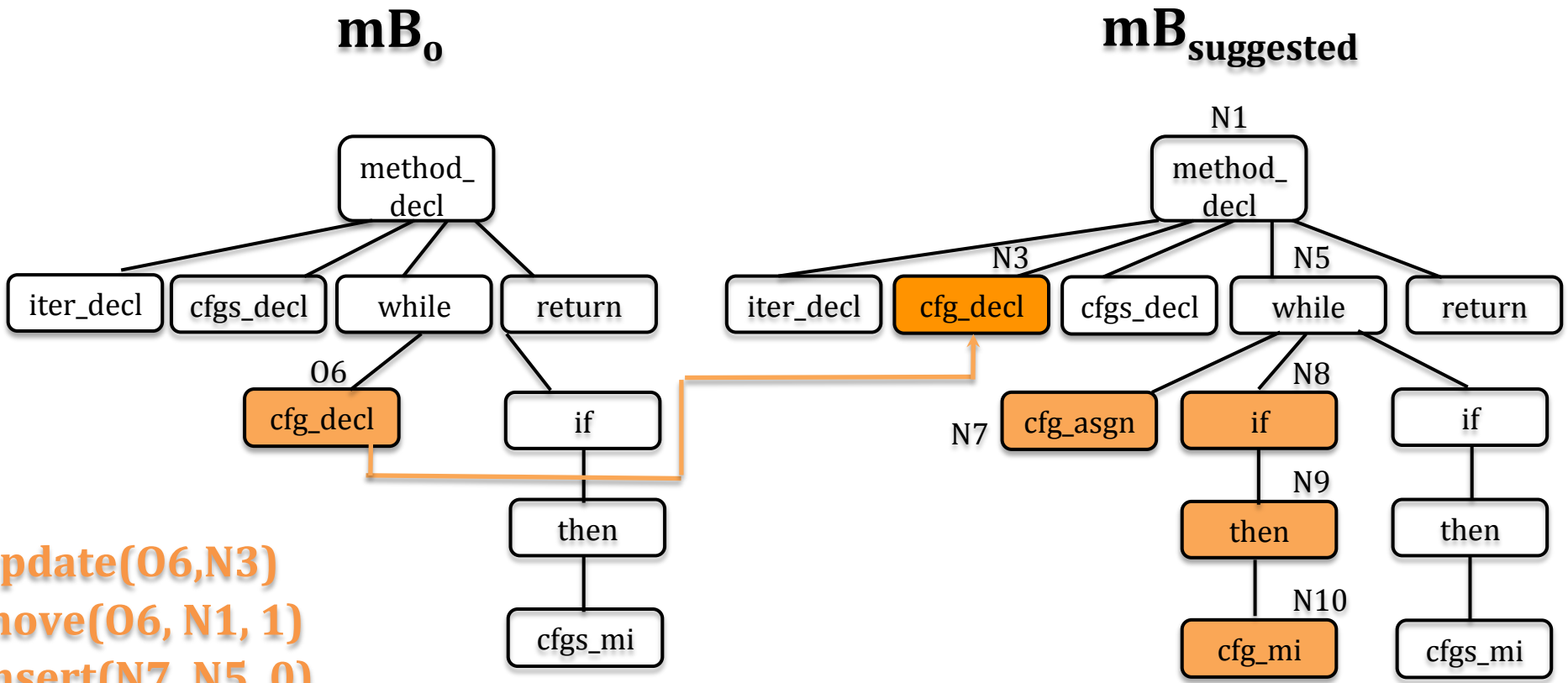
  => cfg = (ILaunchConfiguration) iter.next()

- Edit Position Concretization

  insert (“v1++”, “while(v1<v2)”, 1)

  => insert (“i++”, “while(i<j)”, 2)

Step 6. Concrete Edit Script Application



update(O6,N3)
move(O6, N1, 1)
insert(N7, N5, 0)
insert(N8, N5, 1)
insert(N9, N8, 0)
insert(N10, N9, 0)

Outline

- Program Transformation Generation Approach
- Evaluation on Open Source Projects
- Related Work
- Conclusion

Test Suite

- 56 pairs of exemplar edits from
  - org.eclipse.compare
  - org.eclipse.core.runtime
  - org.debug.core
  - jdt.core
  - jEdit
- Method pairs are at least 40% similar and share at least one common edit

$$similarity(mA,mB) = \frac{2 * |matchingNodes(mA,mB)|}{|mA| + |mB|}$$

Categorization of Edits in the Test Suite

	Single Node	Multiple Nodes	
		Contiguous	Non-contiguous
Identical	**SI**	**CI**	**NI**
	7	7	11
Abstract	**SA**	**CA**	**NA**
	7	12	12

Example: Non-contiguous Abstract Edits

A_{old} to A_{new}

```
  public IAcitionBars getActionBars(){
+ IActionBars actionBars =
    fContainer.getActionBars();
- if (fContainer == null) {
+ if (actionBars == null && !
fContainerProvided){
       return
    Utilities.findActionBars(fComposite
    );
  }
-   return fContainer.getActionBars();
+   return actionBars;
```

B_{old} to B_{new}

```
  public IServiceLocator
    getServiceLocator(){
+ IServiceLocator serviceLocator =
    fContainer.getServiceLocator();
- if (fContainer == null) {
+ if (serviceLocator == null && !
fContainerProvided){
     return
    Utilities.findSite(fComposite);
    }
- return fContainer.getServiceLocator();
+ return serviceLocator;
```

RQ1: Coverage, Accuracy and Similarity

Coverage: What percentage of examples can Sydit match the context and produce some edits?

Accuracy: What percentage of examples can Sydit apply edits correctly?

Similarity: How similar is Sydit-generated version to developer-generated version?

RQ1: Coverage, Accuracy and Similarity

	Single Node	Multiple Nodes	
		Contiguous	Non-contiguous
Identical			
examples	7	7	11
coverage	71%	100%	73%
accuracy	71%	100%	73%
similarity	100%	100%	100%
Abstract			
examples	7	12	12
coverage	100%	75%	83%
accuracy	86%	50%	58%
similarity	86%	95%	95%
Coverage		82% (46/56)	
Accuracy		70% (39/56)	
Similarity		96% (46)	

Sydit creates edits for 82% of examples, produces correct edits for 70%, & with 96% similarity to developer's edits

RQ2: Context Characterization

	% coverage	% accuracy	% similarity
Varying the number of dependence hops			
k=1	79%	66%	95%
k=2	75%	63%	95%
k=3	75%	63%	95%
Varying the abstraction settings			
abstract V T M	82%	70%	96%
abstract V	66%	55%	55%
abstract T	66%	55%	55%
abstract M	80%	68%	96%
no abstraction	66%	55%	55%
Varying the upstream and downstream dependence settings			
all (k=1)	79%	66%	95%
containment only	84%	68%	90%
upstream only (k=1)	82%	70%	96%

Related Work

- Program differencing
- Refactoring
- Source transformation
- Simultaneous text editing [Miller et al.]
- Generic patch inference [Andersen & Lawall]

Conclusion

Sydit ***automates systematic edits*** by generating transformations from an example and applies them to different contexts with

82% coverage 70% accuracy, and 96% similarity

Future Work

- Select target methods automatically
- Present proposed edits in a *diff*-style view for programmers to preview
- Integrate with ***automated compilation and testing***

Thank You !