Change Type Extraction with ChangeDistiller

Beat Fluri, Harald Gall
University of Zurich, Switzerland

s.e.a.l.
software evolution & architecture lab

University of Zurich
Department of Informatics

Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();
```

CVS log: “lines: +2 -4”

Change Analysis

CVS diff

```
1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();
```

CVS log: “lines: +2 -4”

- 3 Body changes

Change Analysis

CVS diff

```
1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();
```

CVS log: “lines: +2 -4”

- 3 Body changes
  - 2 Statement parent changes

Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();
```

- 3 Body changes
  - 2 Statement parent changes
  - 1 Statement delete

CVS log: “lines: +2 -4”

Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();
```

CVS log: “lines: +2 -4”

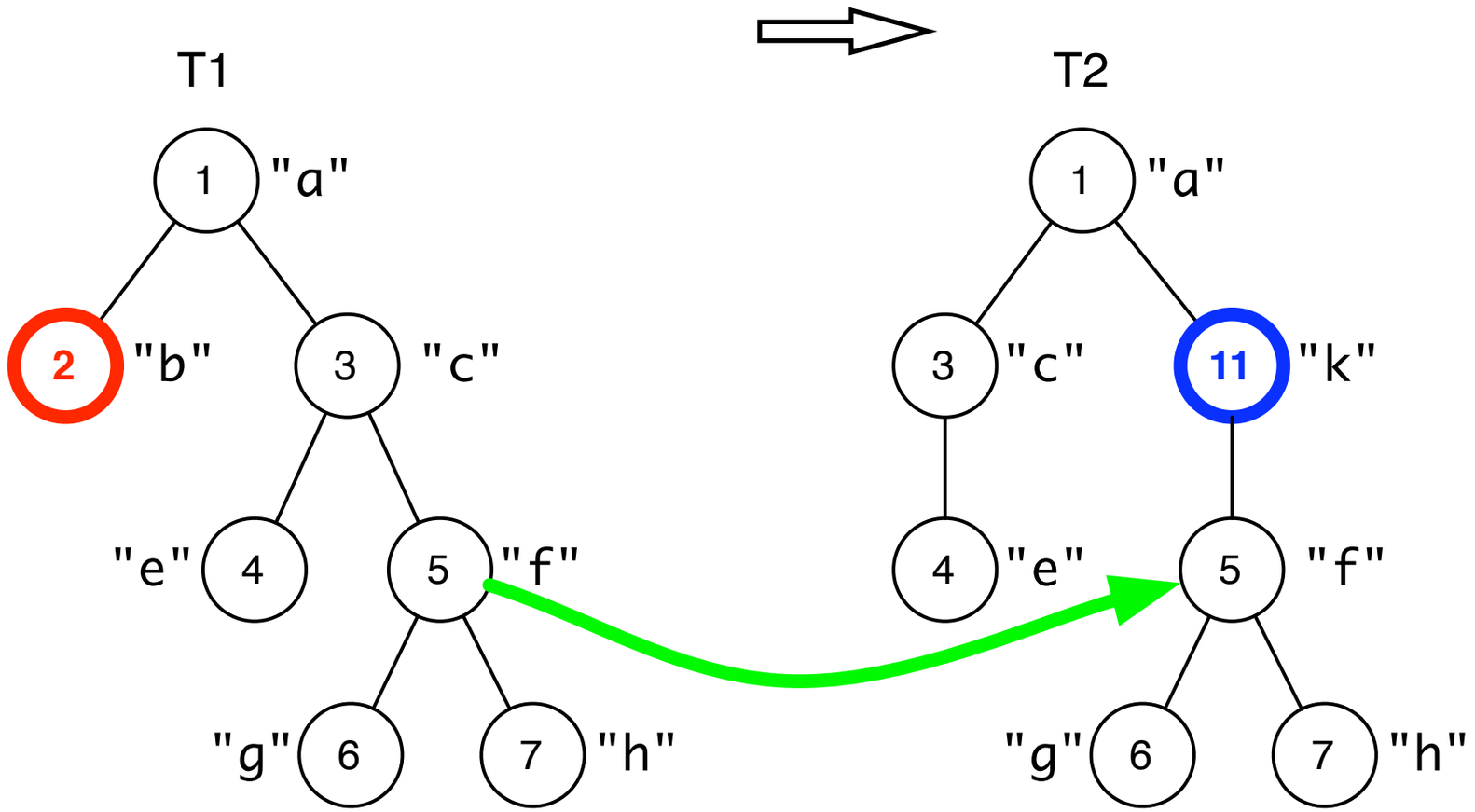
- 3 Body changes
  - 2 Statement parent changes
  - 1 Statement delete
- Change significance

Outline

- Source code changes using abstract syntax trees (AST)
- Taxonomy of source code changes
- ChangeDistiller tool

Tree Differencing

S. S. Chawathe et al., *Change Detection in Hierarchically Structured Data*, SIGMOD 1996



INS((11,"k"),1,3) MOV(5,11,1) DEL(2)

Source Code Changes using ASTs

- Using tree differencing, we can determine

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Source Code Changes using ASTs

- Using tree differencing, we can determine
  - enclosing entity (root node)
  - kind of statement which changed (node information)
  - kind of change (tree edit operation)

```
public void method(D d) {
    if (d != null) {
        d.foo();
        d.bar();
    }
}
```

```
public void method(D d) {
    d.foo();
    d.bar();
}
```

Tree Differencing

- Using tree differencing algorithm on an AST
- Problems
  - Tree differencing algorithm expects a general tree data structure
  - AST implementation not uniform tree structure
    - Children of a AST-Node are not accessible uniformly (*e.g.*, if-statement: `getThenStatement()` instead of `getChildren()`)

Transforming an AST

- Transform an AST in a uniform tree data structure
- Using labeled and valued node
  - Label: number representing statement kind

Transforming an AST

```
if (d != null) {
   d.foo();
   d.bar();
}
```

Transforming an AST

```
if (d != null) {
    d.foo();
    d.bar();
}
```

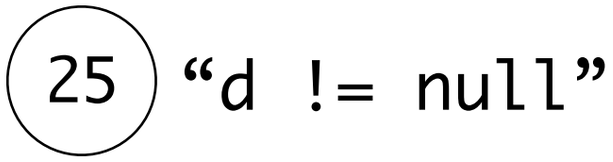
Transforming an AST

```
if (d != null) {
    d.foo();
    d.bar();
}
```

25

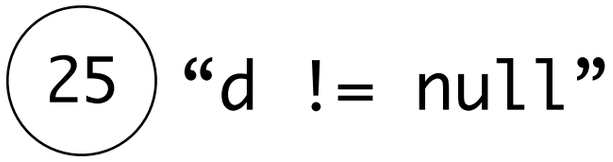
Transforming an AST

```
if (d != null) {
    d.foo();
    d.bar();
}
```



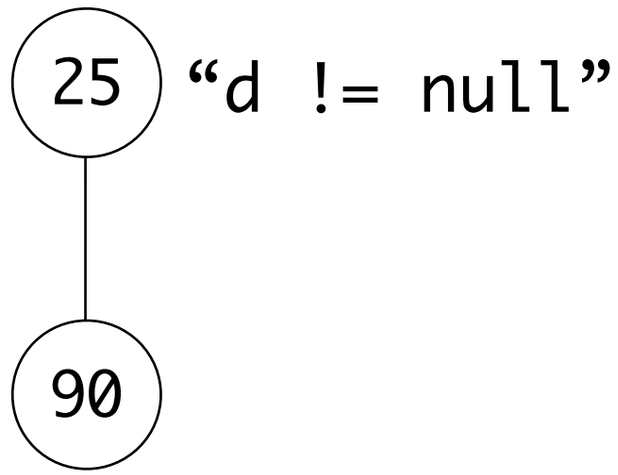
Transforming an AST

```
if (d != null) {
   d.foo();
   d.bar();
}
```



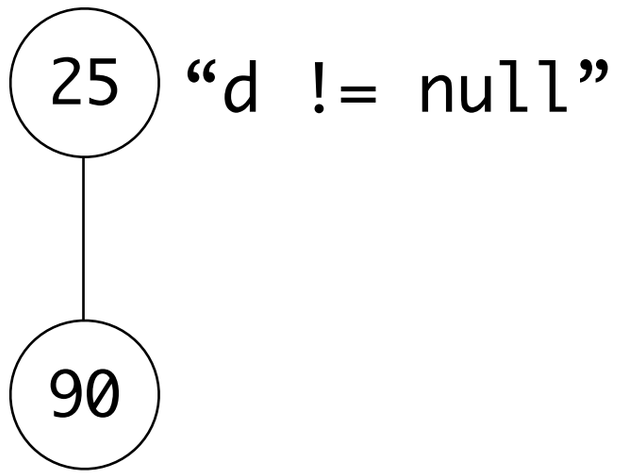
Transforming an AST

```
if (d != null) {
   d.foo();
   d.bar();
}
```



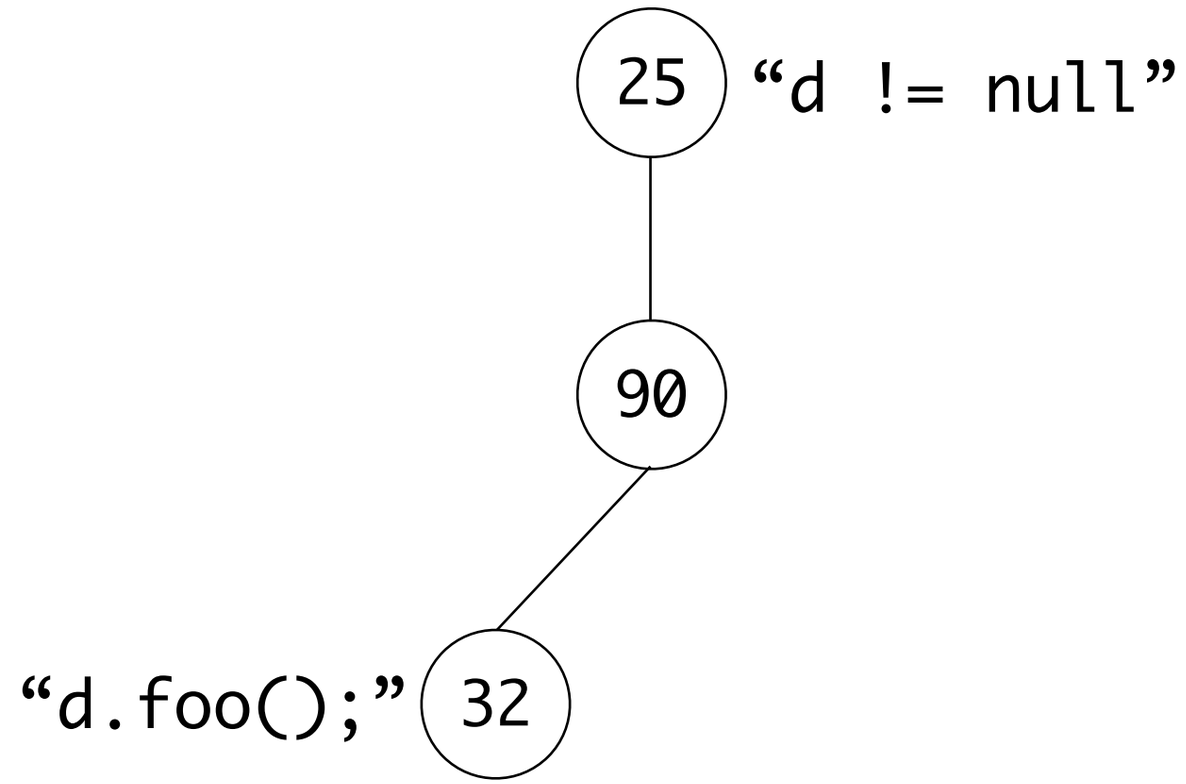
Transforming an AST

```
if (d != null) {
    d.foo();
    d.bar();
}
```



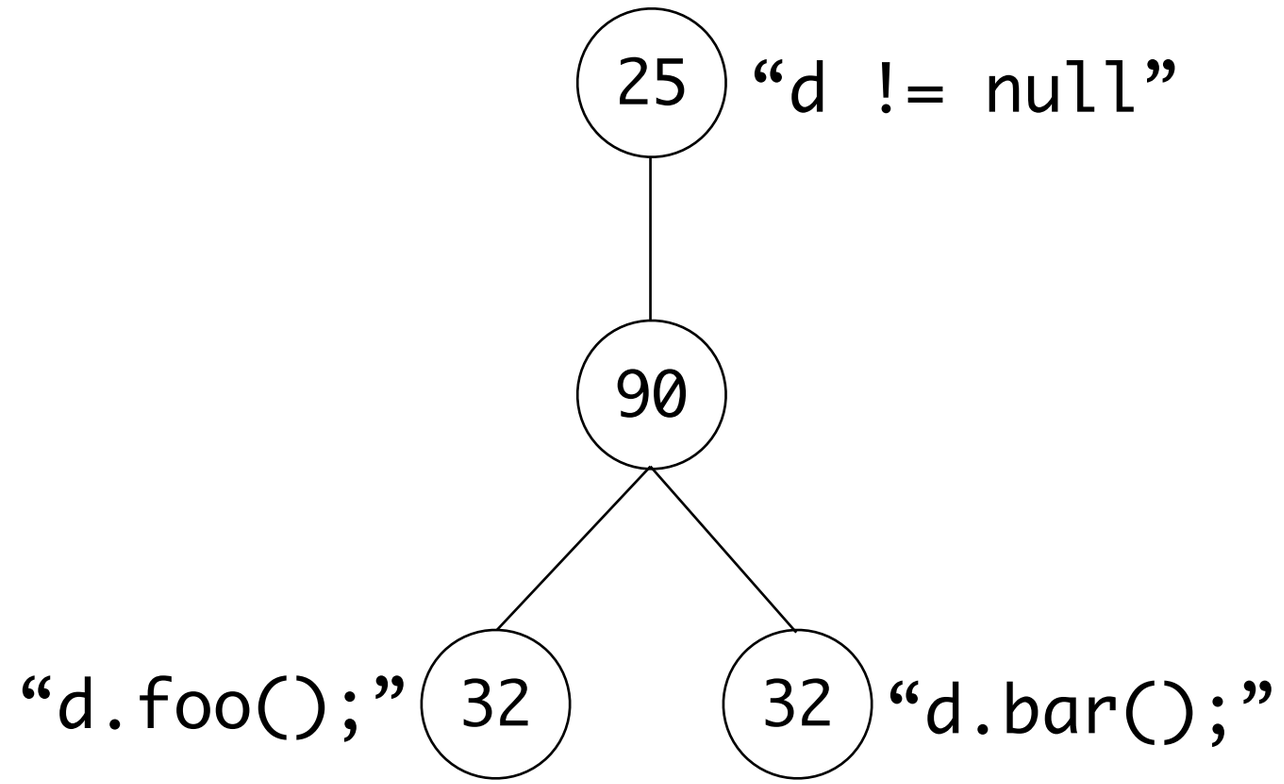
Transforming an AST

```
if (d != null) {
    d.foo();
    d.bar();
}
```



Transforming an AST

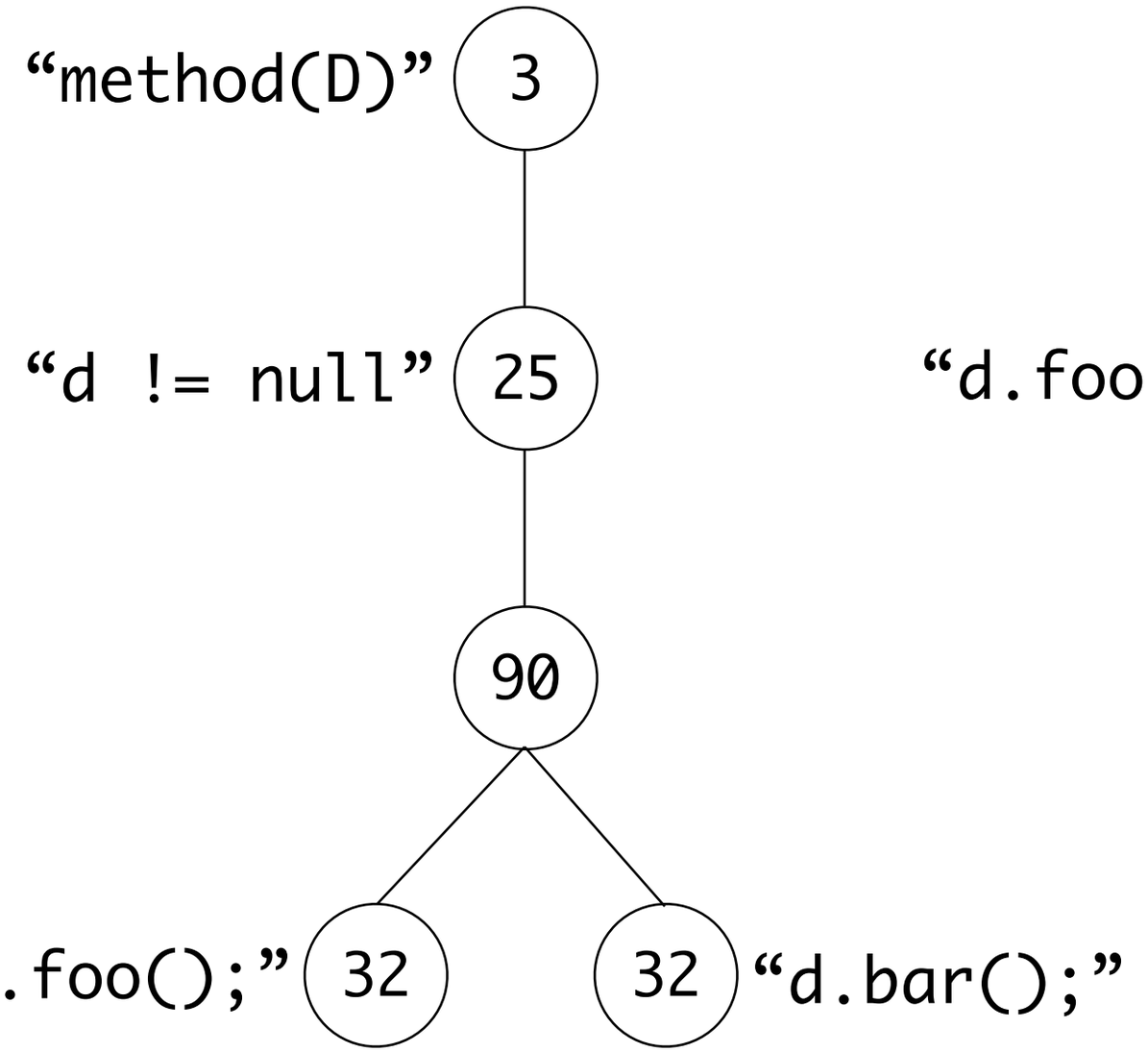
```
if (d != null) {
    d.foo();
    d.bar();
}
```

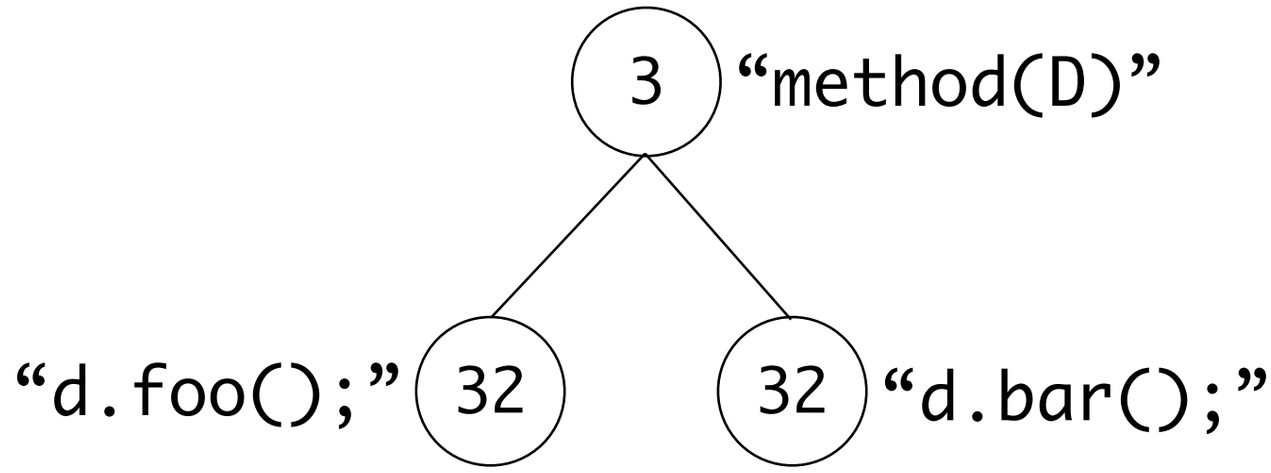


Tree Differencing

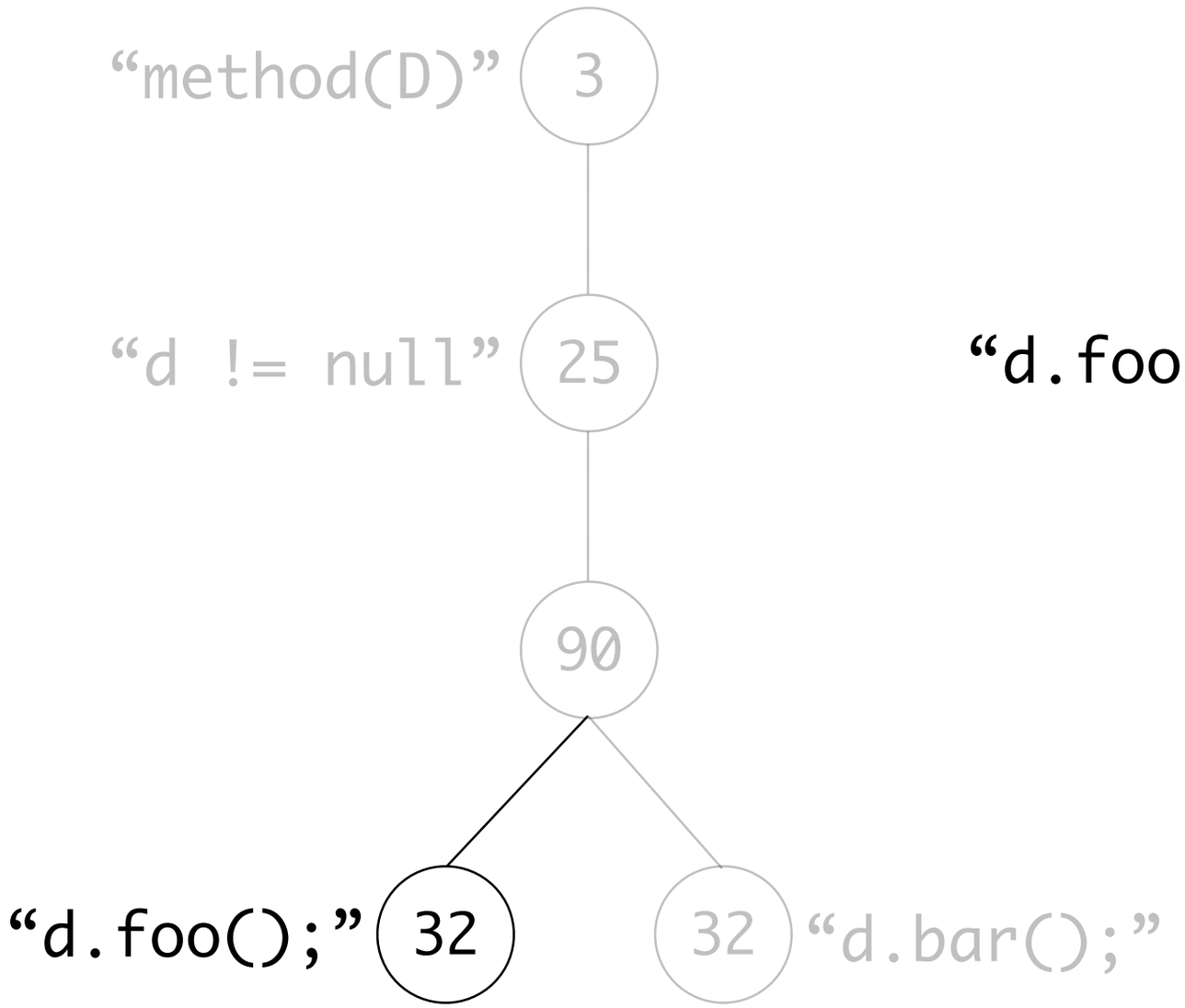
- Phase 1: building a matching set between nodes
- Phase 2: building an edit script transforming left into right tree, based on matching set

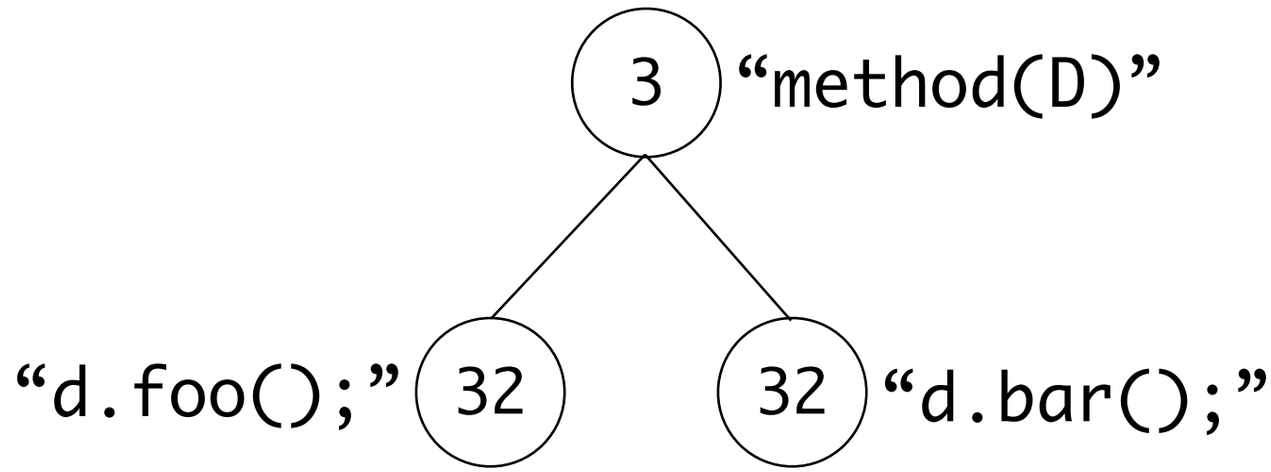
Building a Matching Set



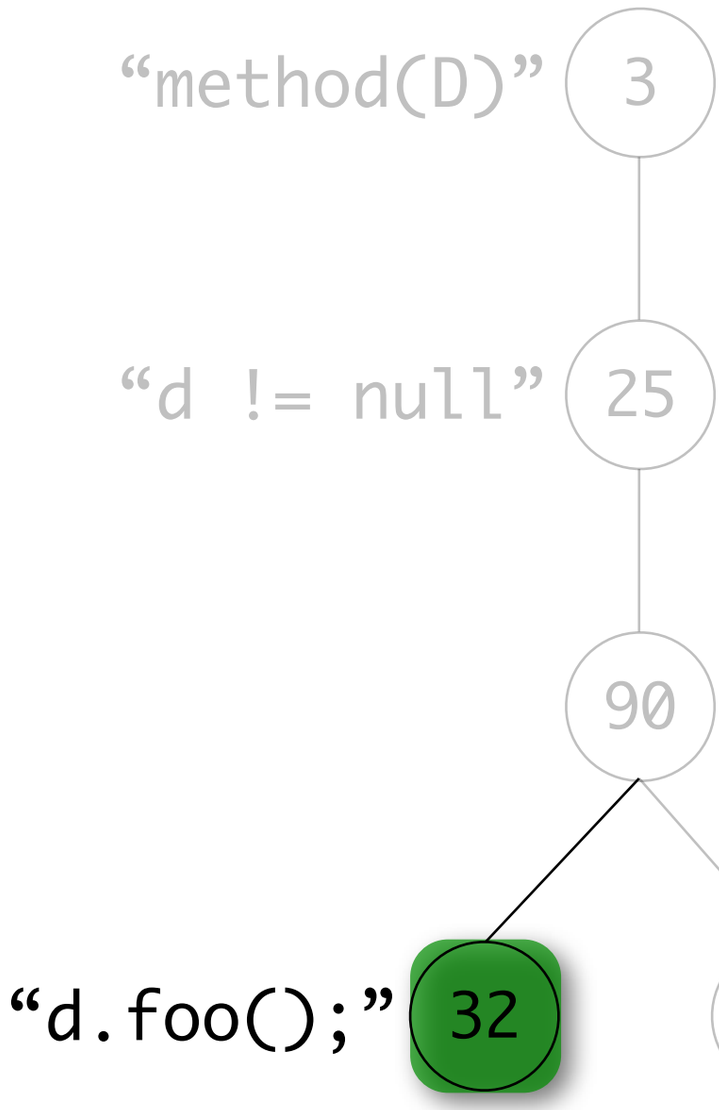


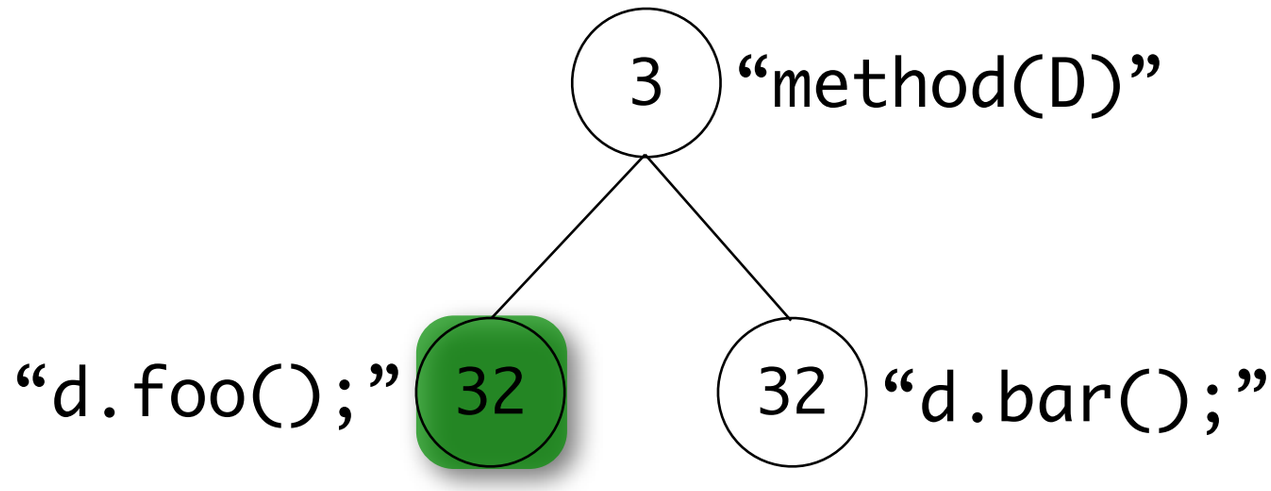
Building a Matching Set



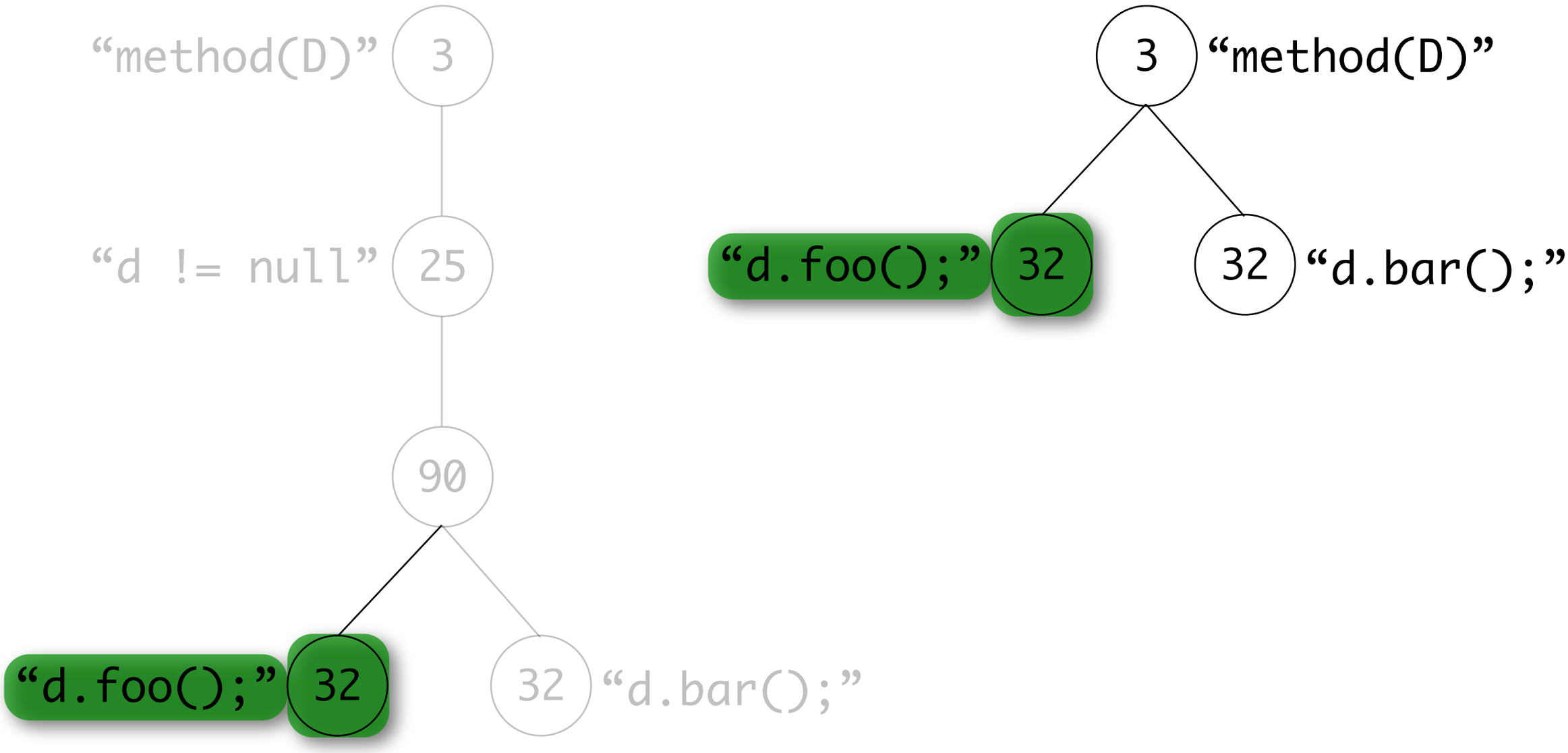


Building a Matching Set

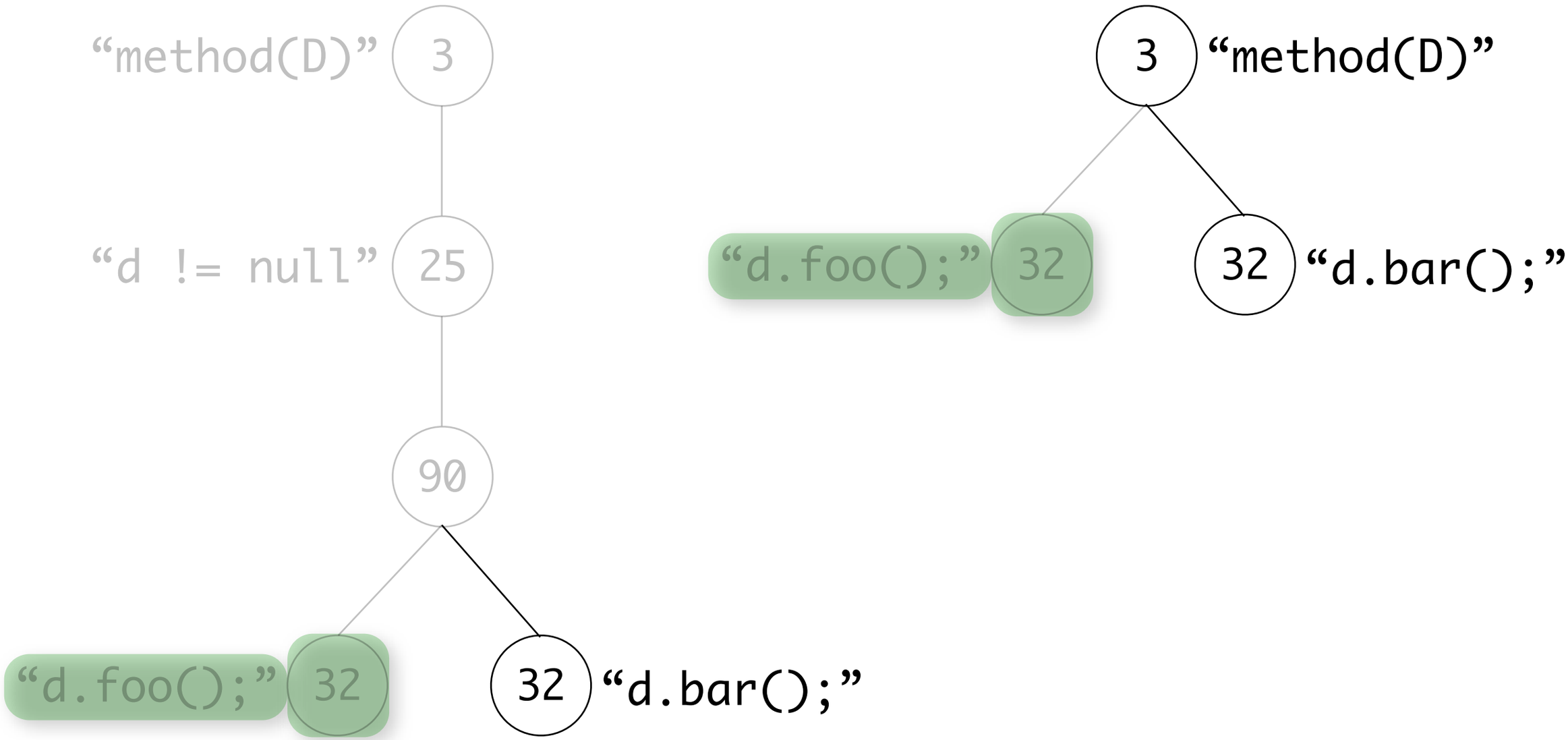




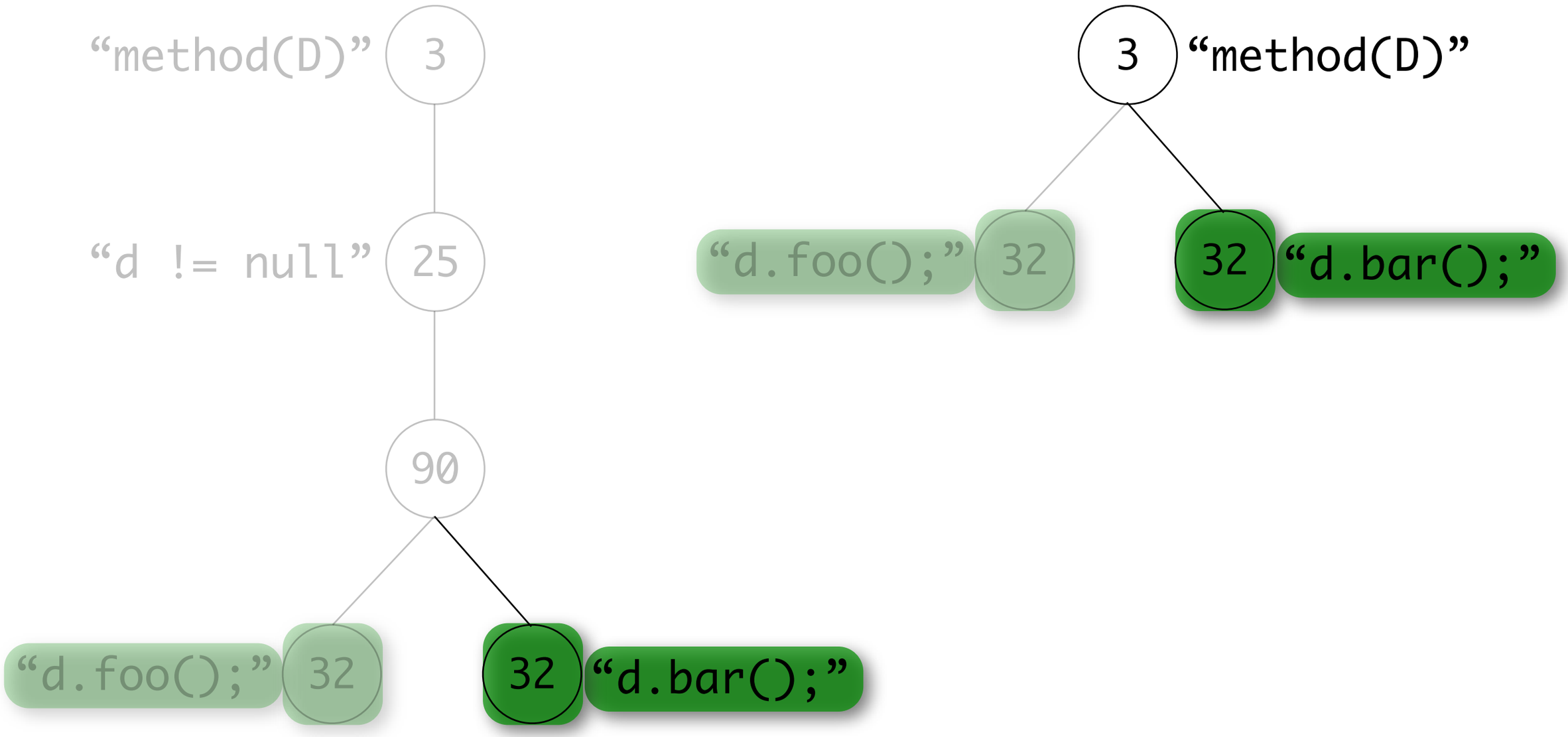
Building a Matching Set



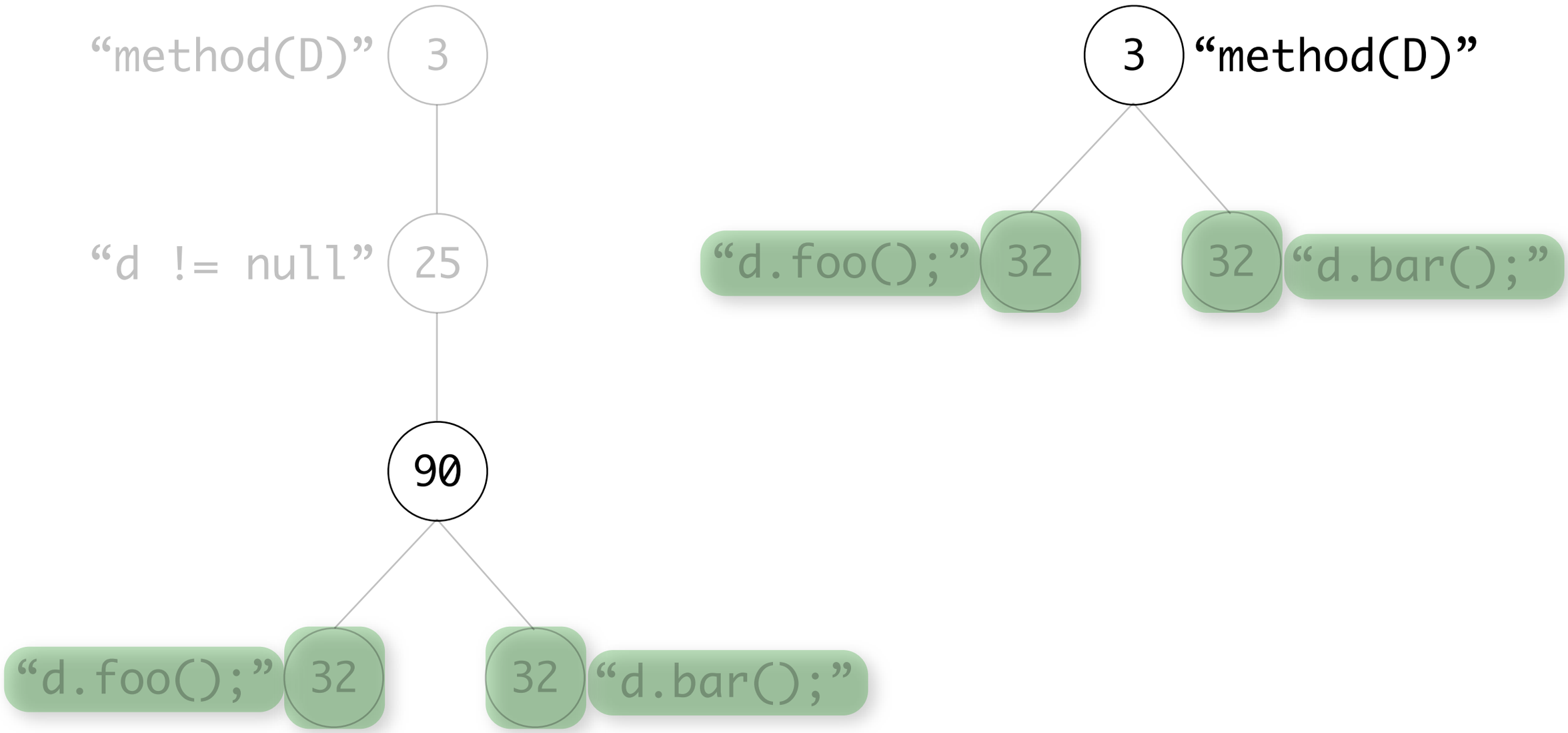
Building a Matching Set



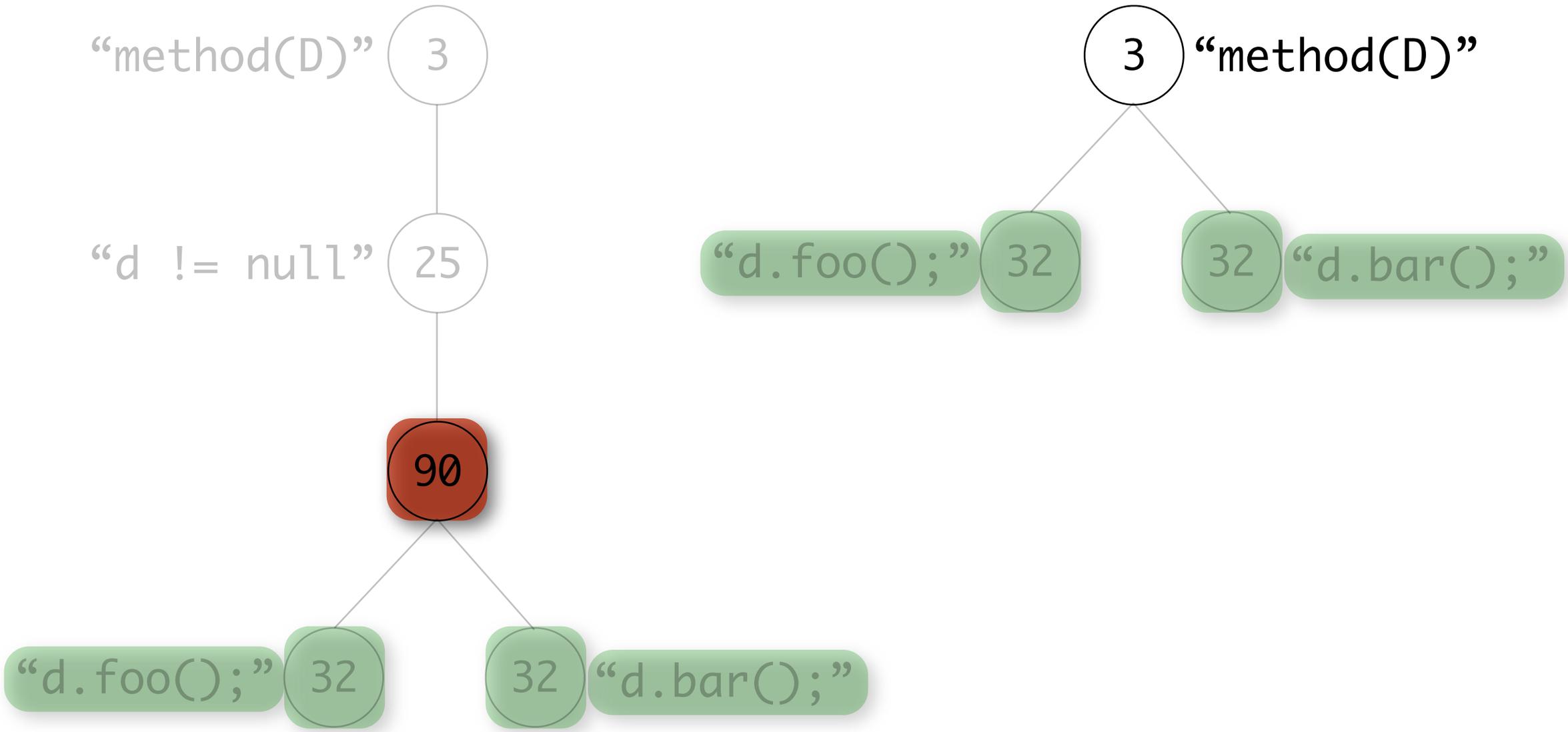
Building a Matching Set



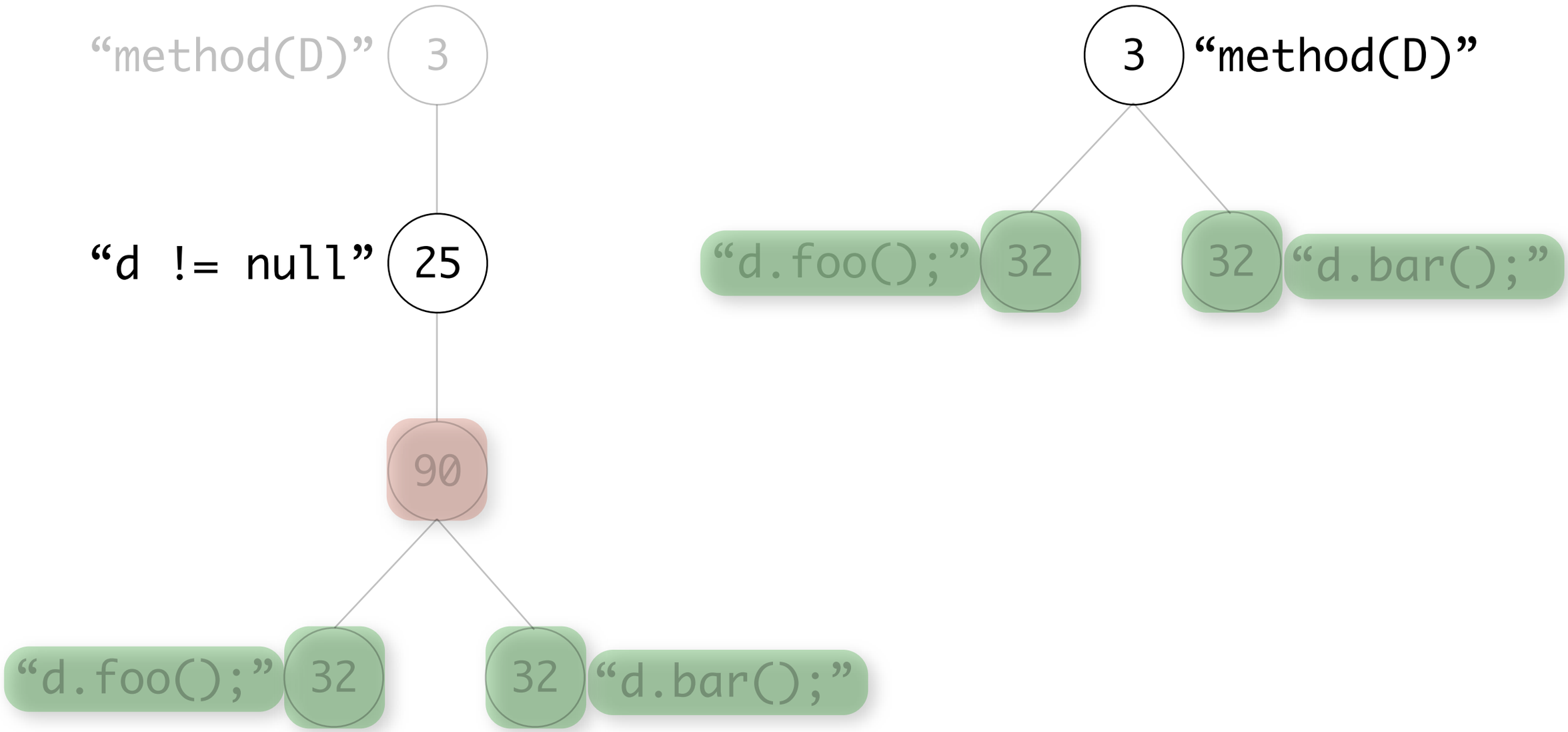
Building a Matching Set



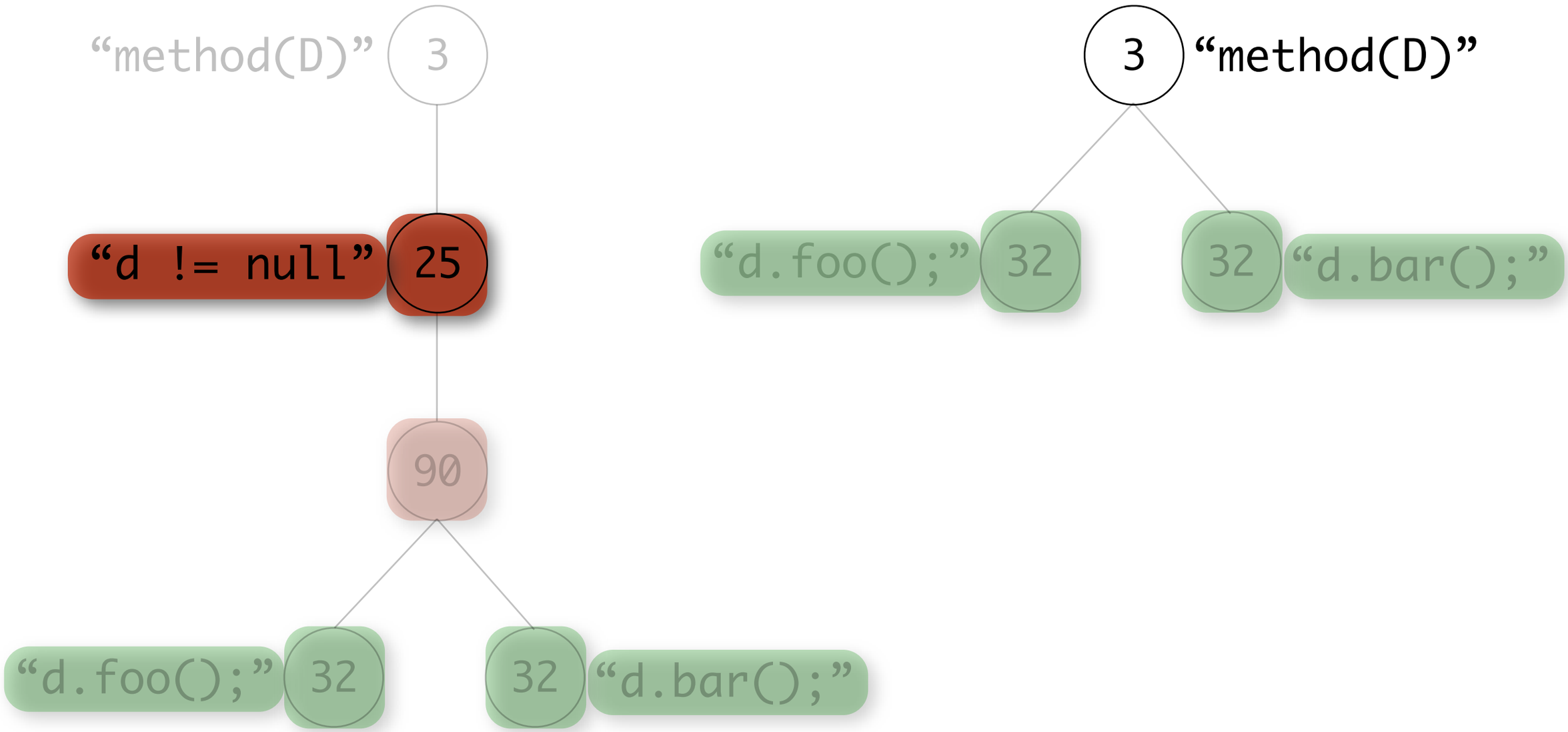
Building a Matching Set



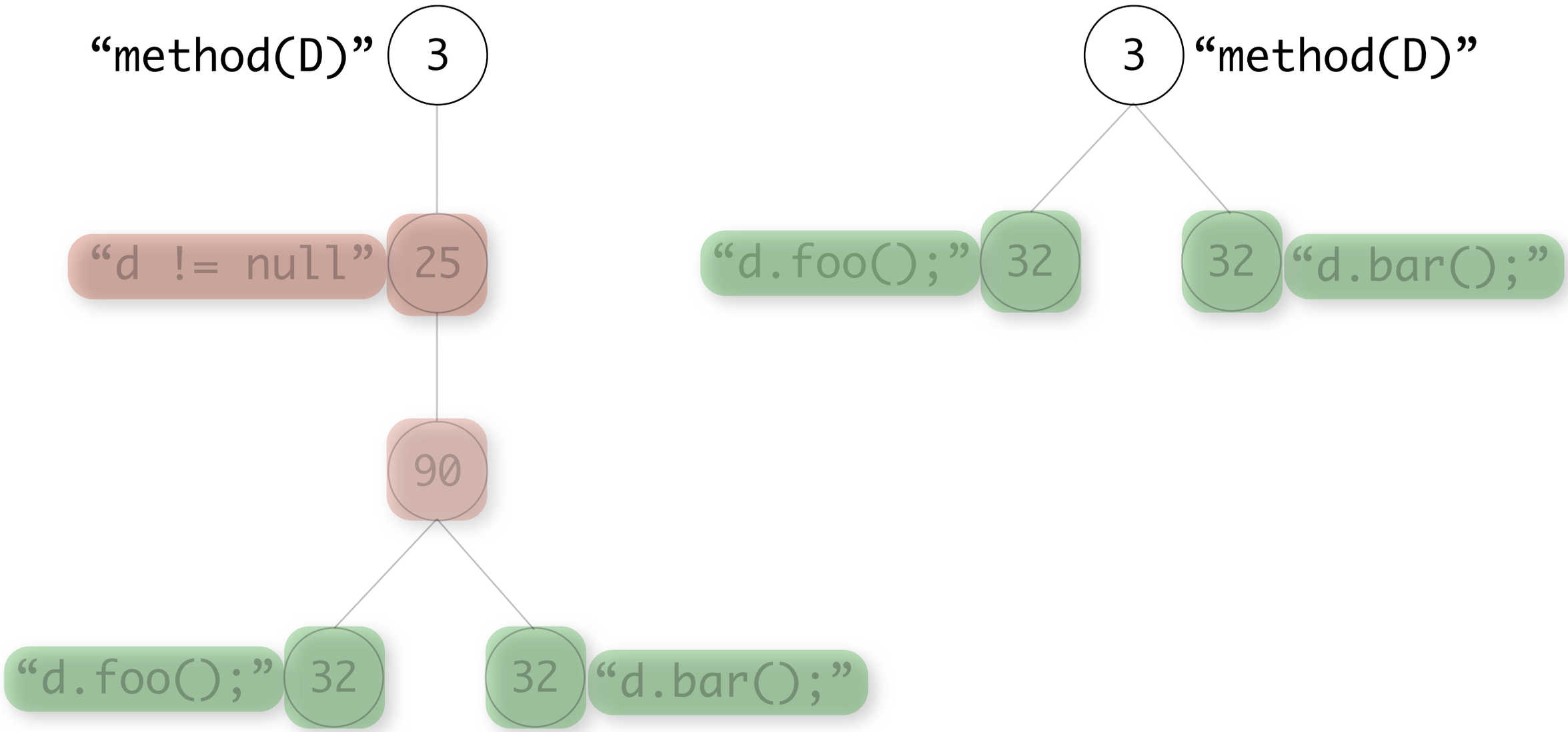
Building a Matching Set



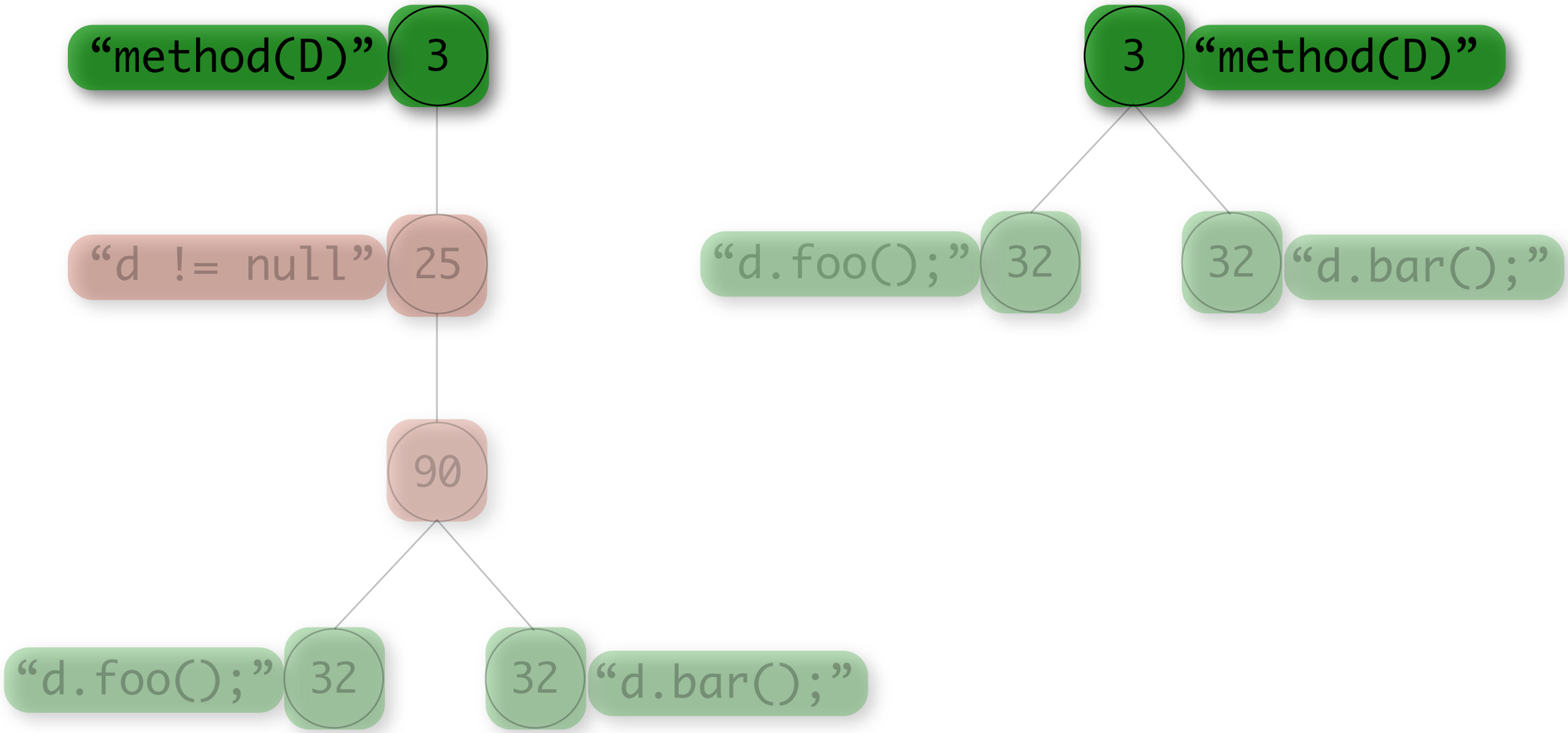
Building a Matching Set



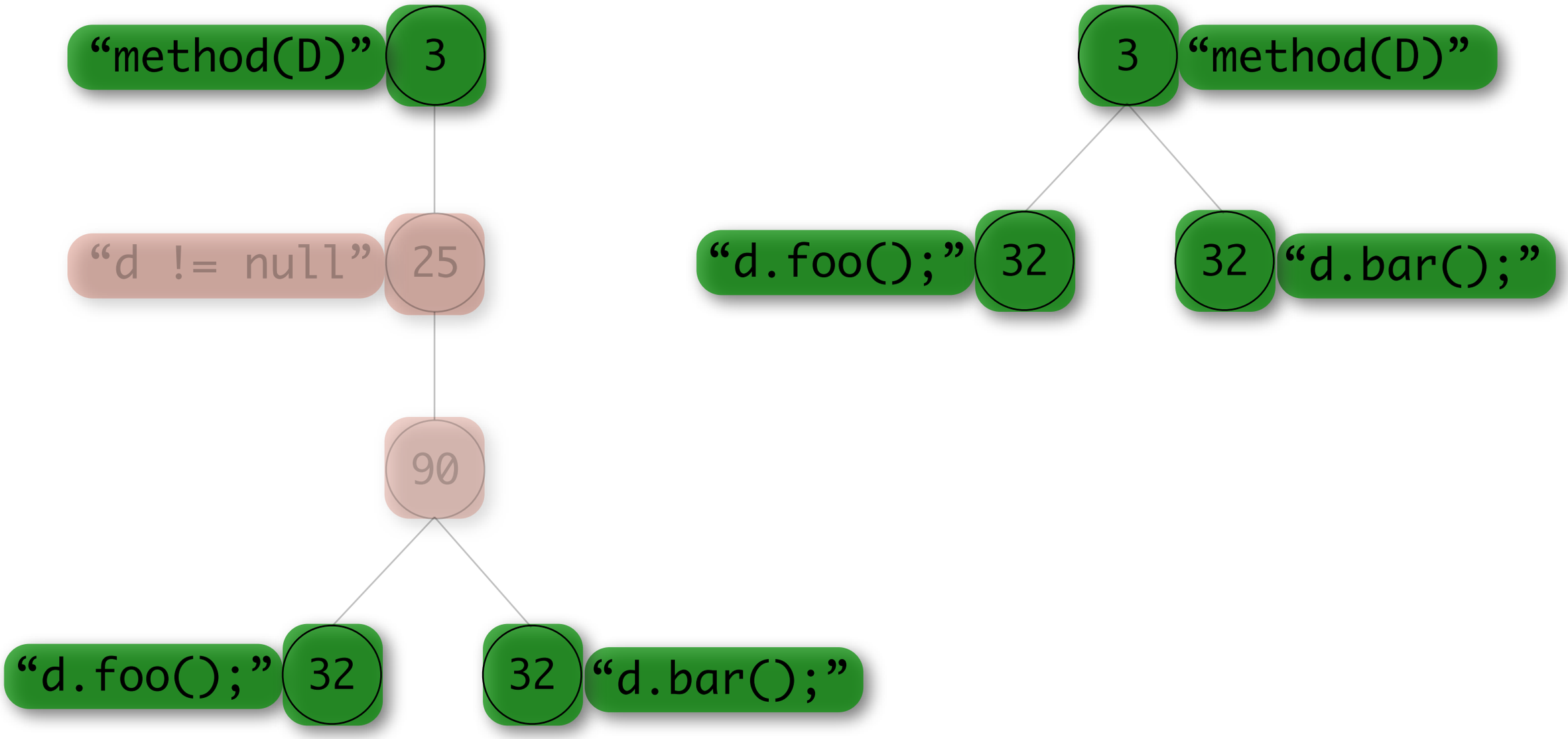
Building a Matching Set



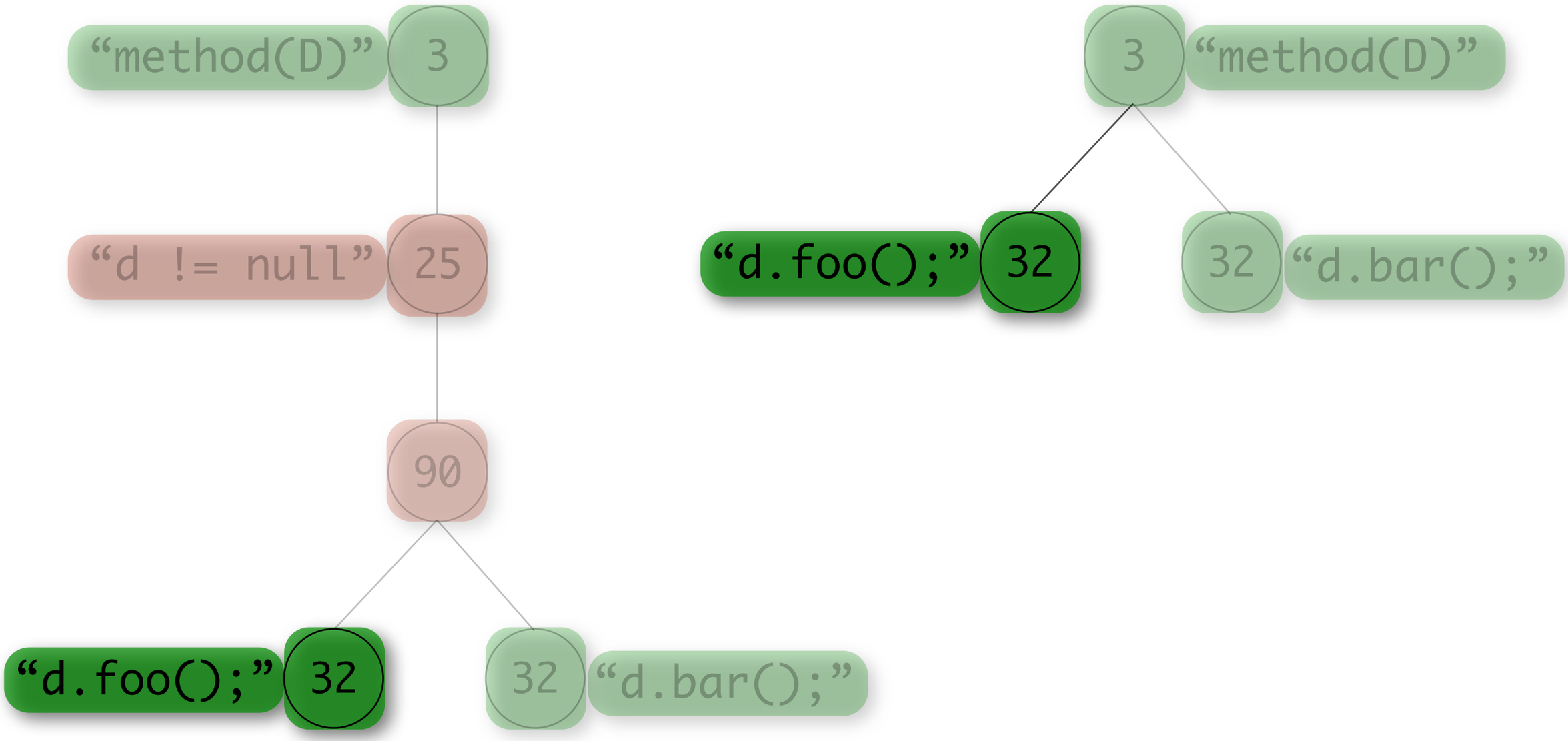
Building a Matching Set



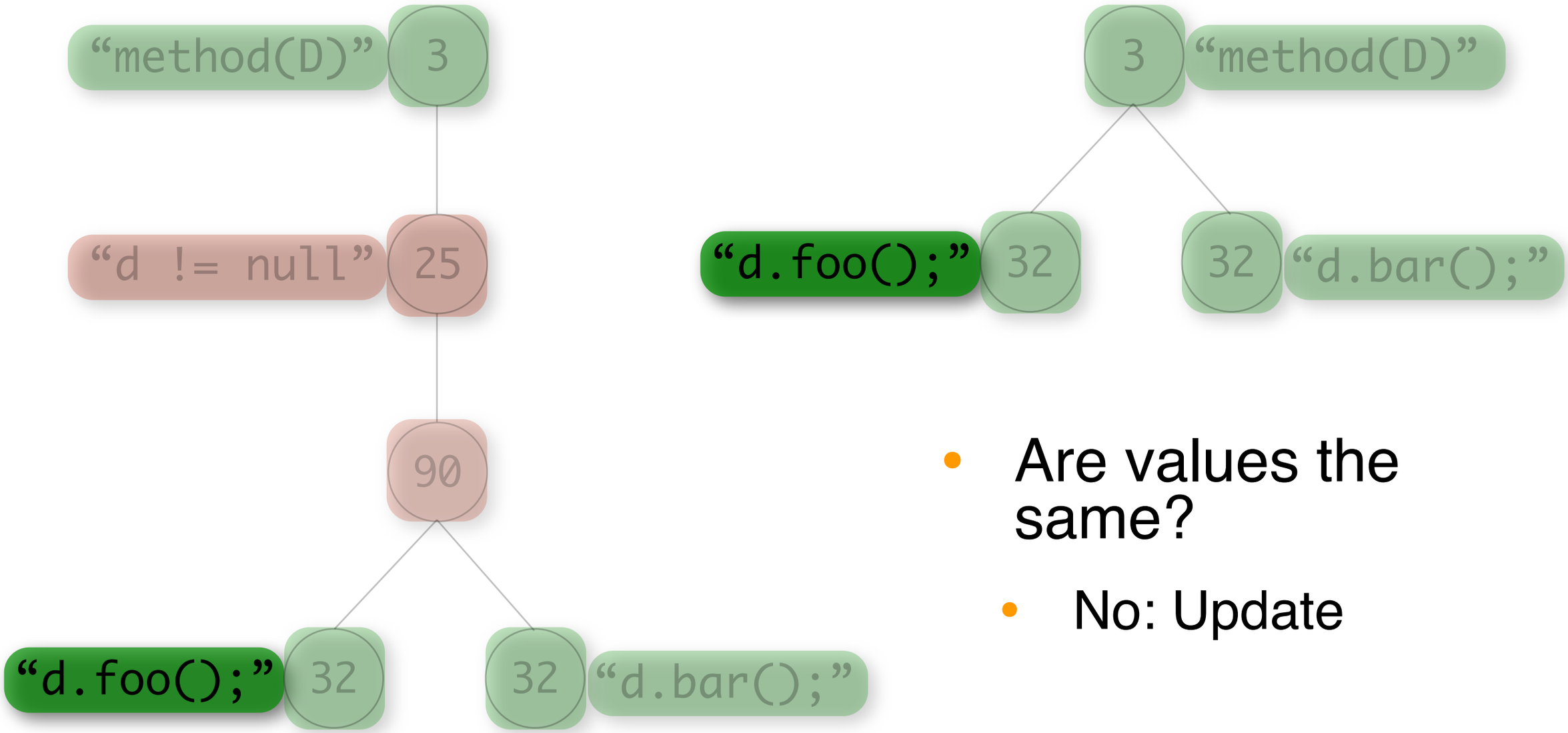
Building a Matching Set



Building the Edit Script

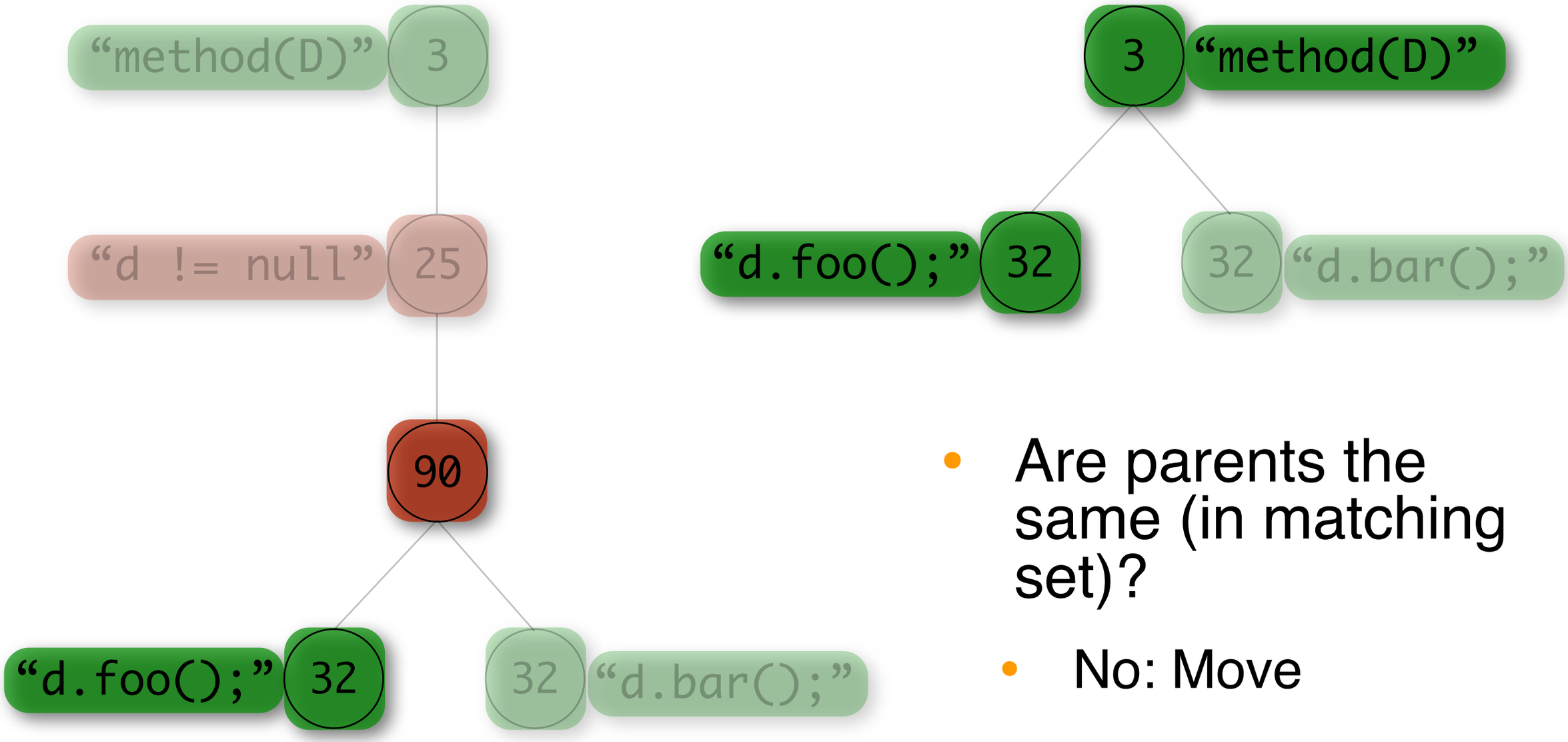


Building the Edit Script



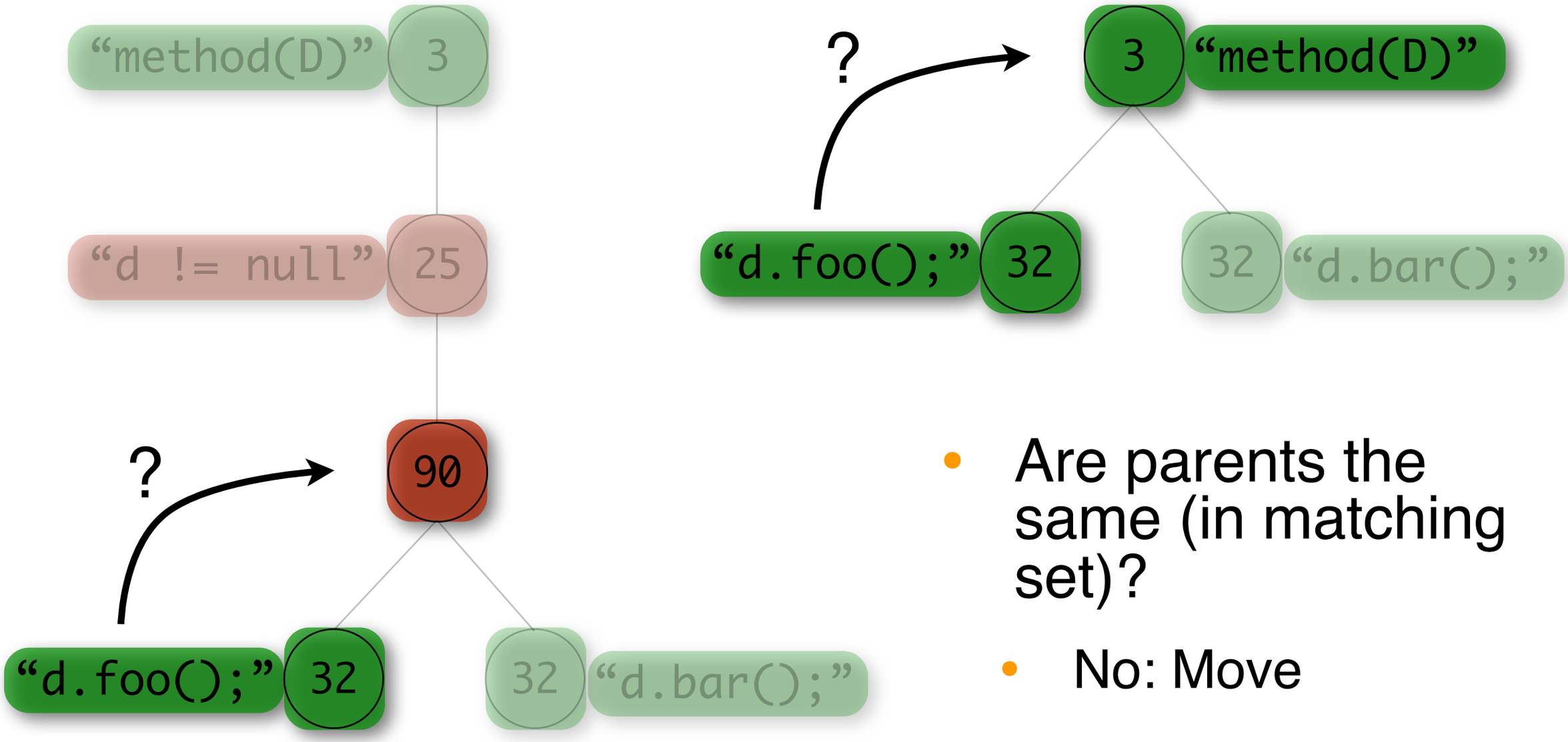
- Are values the same?
  - No: Update

Building the Edit Script



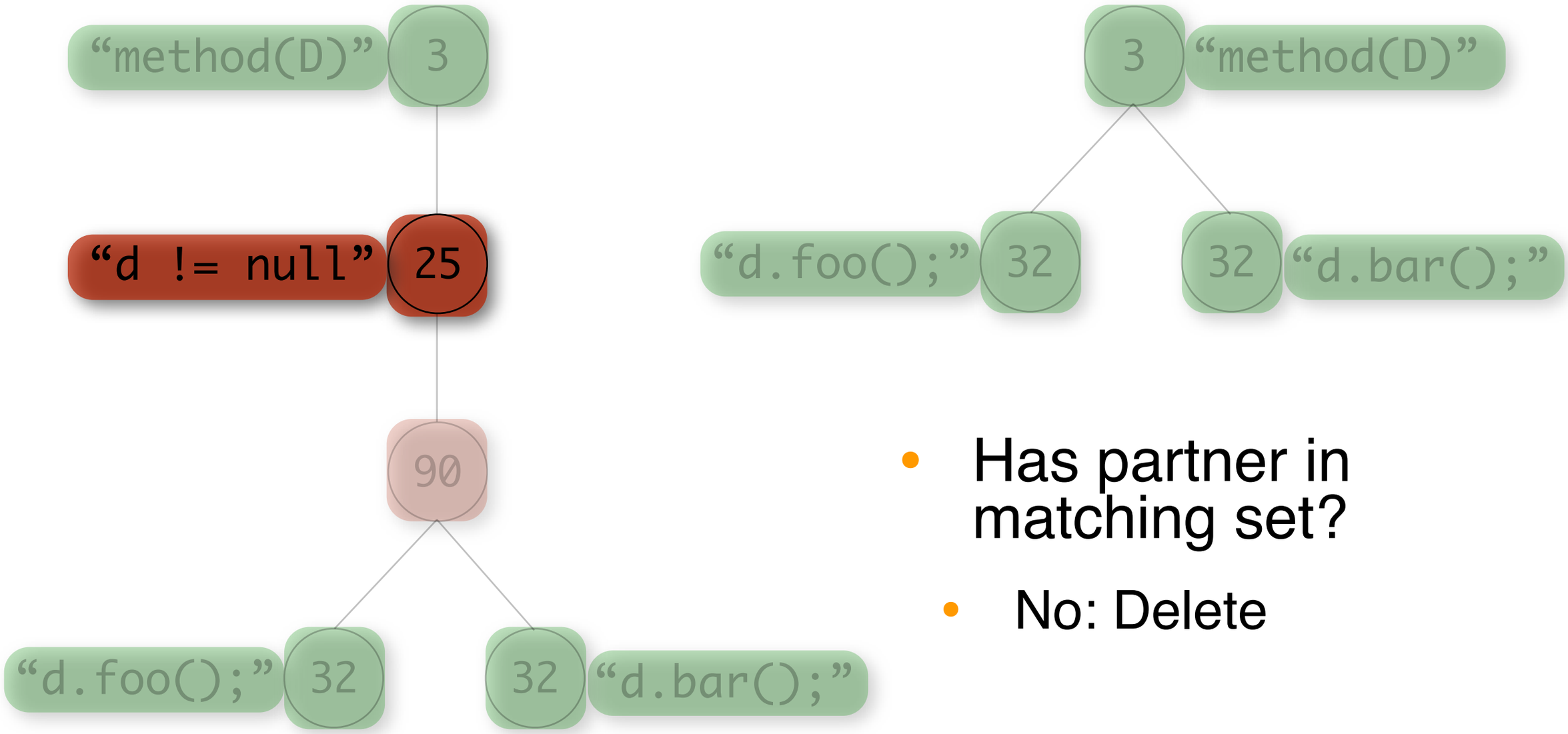
- Are parents the same (in matching set)?
  - No: Move

Building the Edit Script



- Are parents the same (in matching set)?
  - No: Move

Building the Edit Script



- Has partner in matching set?
  - No: Delete

Software Evolution Context

- Taking two ASTs (same class, two subsequent revisions)
- Transforming them into intermediate ASTs
- Applying tree differencing algorithm reporting set of tree edit operations transforming first into second AST
- Using **taxonomy of source code changes** to classify tree edit operations

Taxonomy of Source Code Changes

- Taxonomy classifies a single or a set of tree edit operations into a change type
  - body- or declaration-part change
  - name for the change according to the kind of the operations and involved tree-nodes, *e.g.*,
    - Statement Insert
    - Condition Expression Change
    - Method Renaming

Taxonomy of Source Code Changes

- Taxonomy gives each change type a **change significance level**
  - impact of the change on other source code entities
    - Parameter Renaming vs. Method Renaming
  - whether the change is **functionality-modifying** or **-preserving**
    - Method Renaming vs. Return Type Change
  -

Taxonomy of Source Code Changes

- Four levels of change significance
  - **low**, **medium**, **high**, or **crucial**
  - depending on how strong the impact could be and whether the change is functionality-modifying or -preserving

Examples of Change Types

- Classification of 35 change types
  - body part changes

Additional Object State	low
Condition Expression Change	medium
Removed Functionality	crucial

  - declaration part change

Final Modifier Delete	low
Parameter Renaming	medium
Return Type Update	crucial

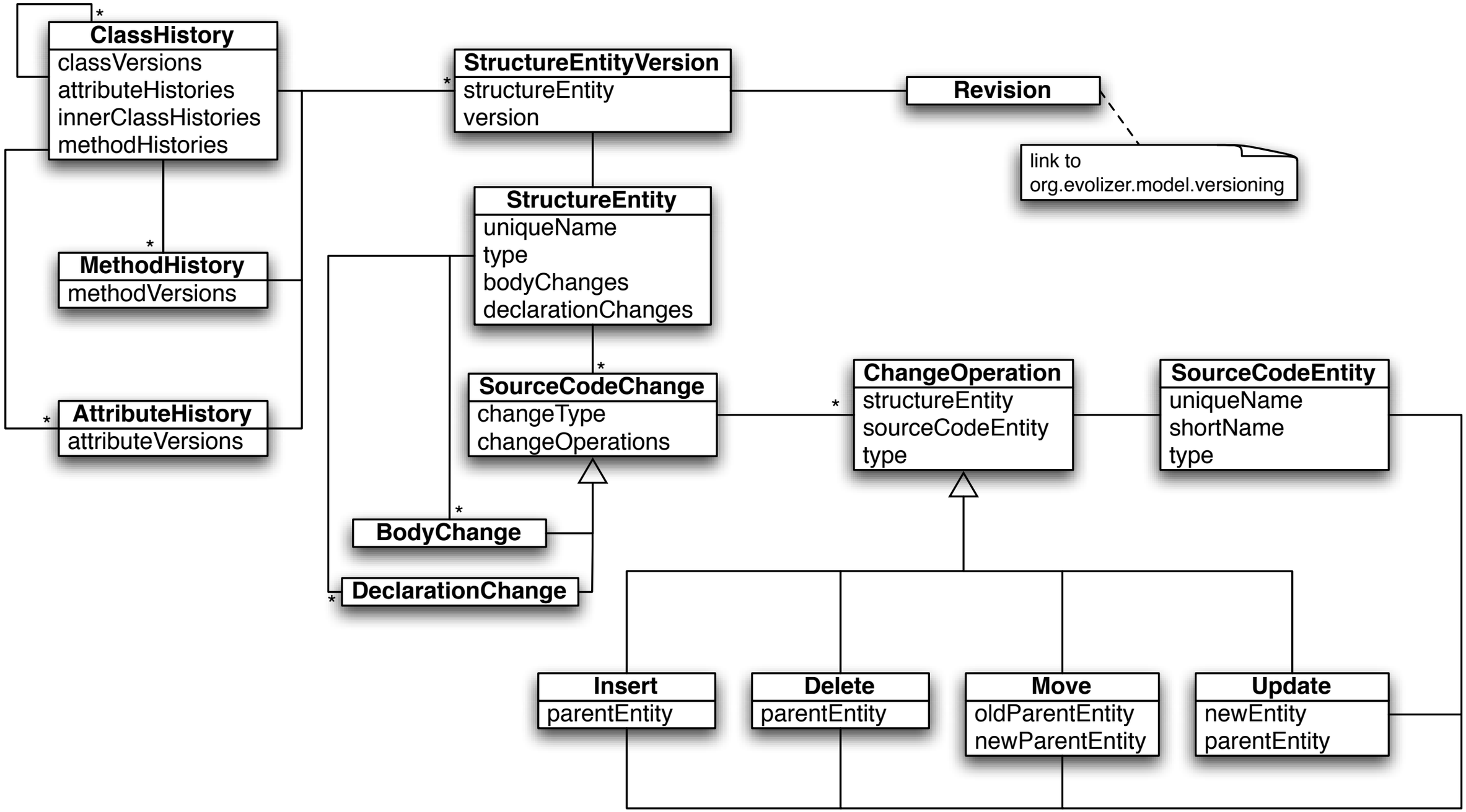
Software Quality and Changes

- Stability of interfaces
- Change propagation analysis
- Documenting (comments) of source code
- Changes due to bug fixes
- Many vs. significant changes

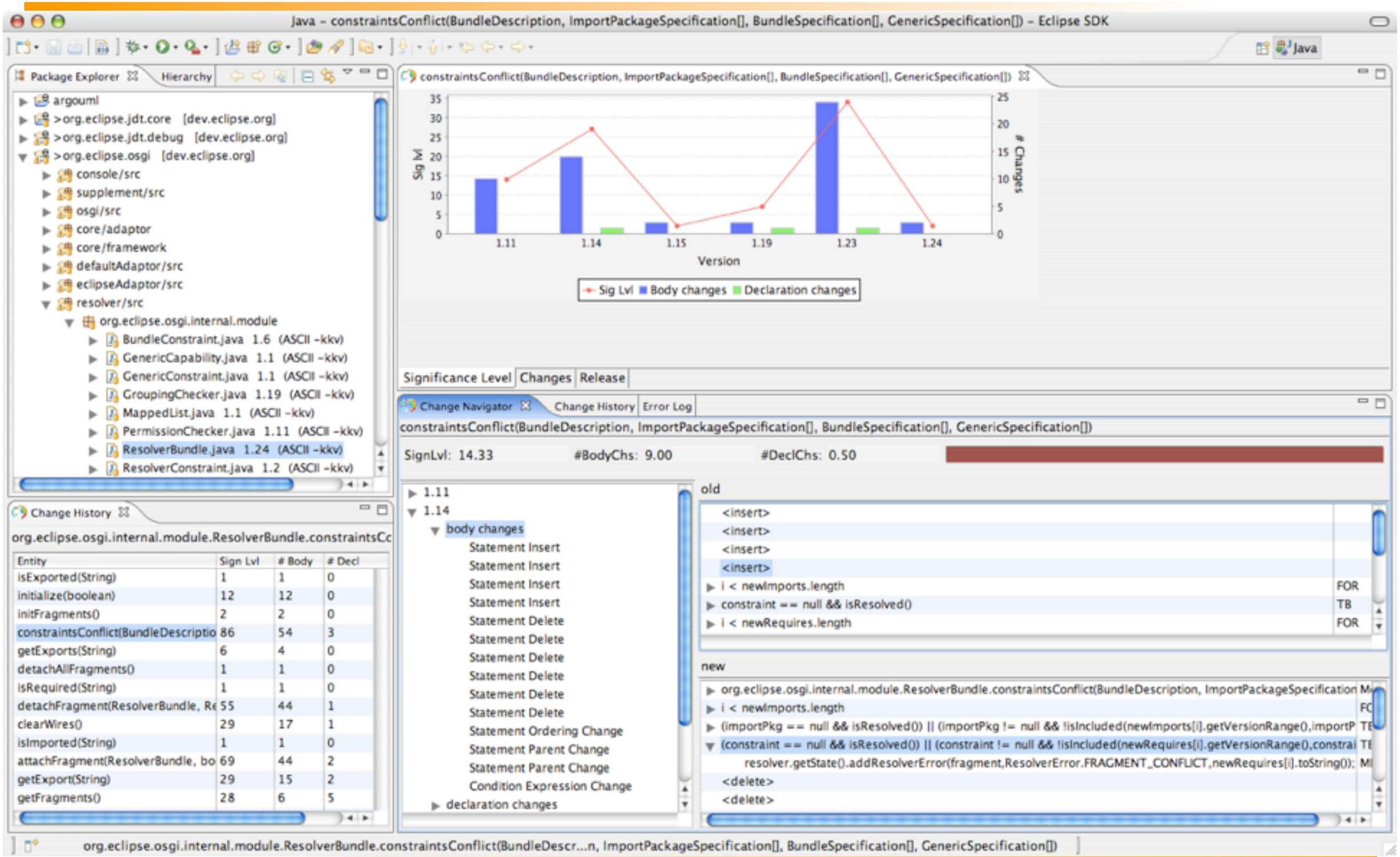
ChangeDistiller

- Implementation as Eclipse plugin
- Use Java Development Tools (JDT)
  - Parser to generate AST
  - AST visitor to generate intermediate tree
- Hibernate (Object Relation Mapper)
  - Object-Oriented model mapped to relational database

ChangeDistiller Model



ChangeDistiller



Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();

CVS log: “lines: +2 -4”
```

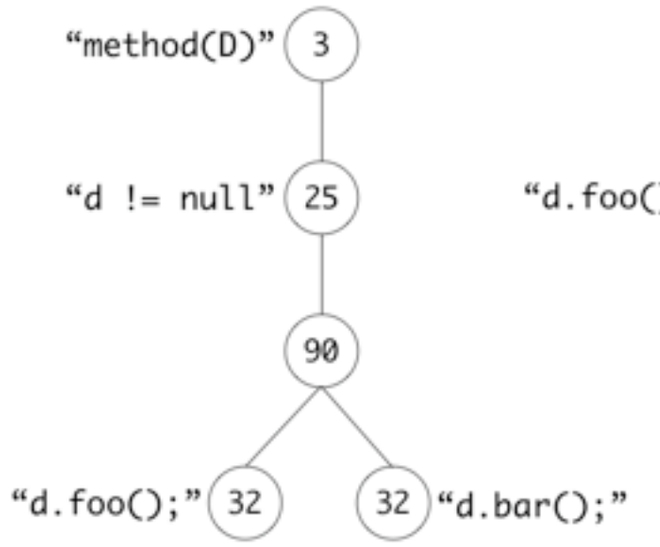
- 3 Body changes
  - 2 Statement parent changes
  - 1 Statement delete
  - Change significance

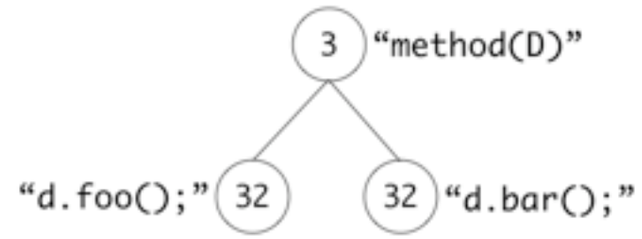
Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();

CVS log: “lines: +2 -4”
```

- 3 Body changes
  - 2 Statement parent changes
  - 1 Statement delete
  - Change significance

Building a Matching Set



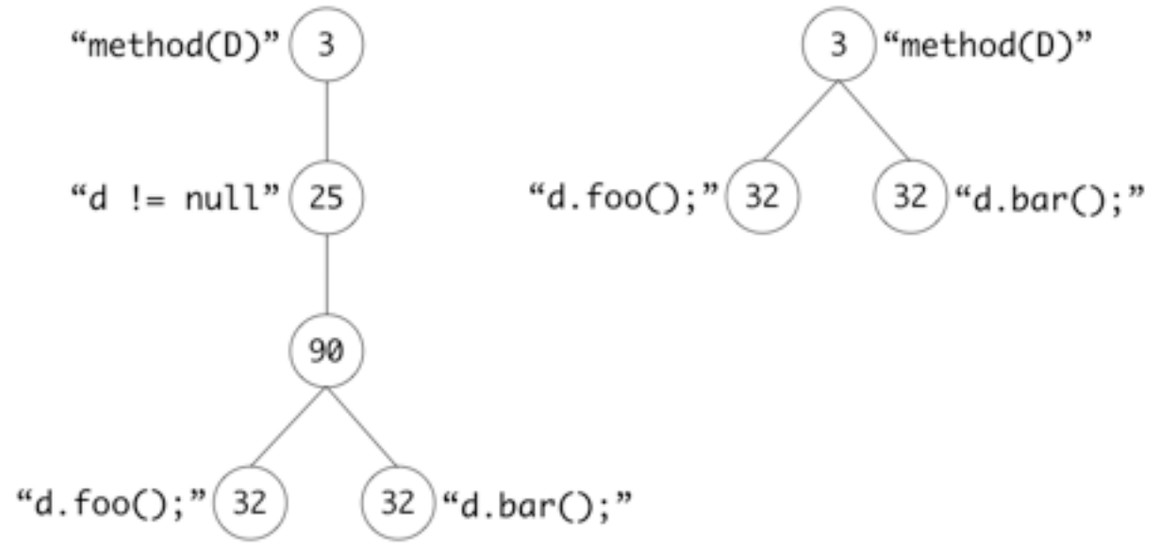


Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();

CVS log: “lines: +2 -4”
```

- 3 Body changes
  - 2 Statement parent changes
  - 1 Statement delete
  - Change significance

Building a Matching Set



Taxonomy of Source Code Changes

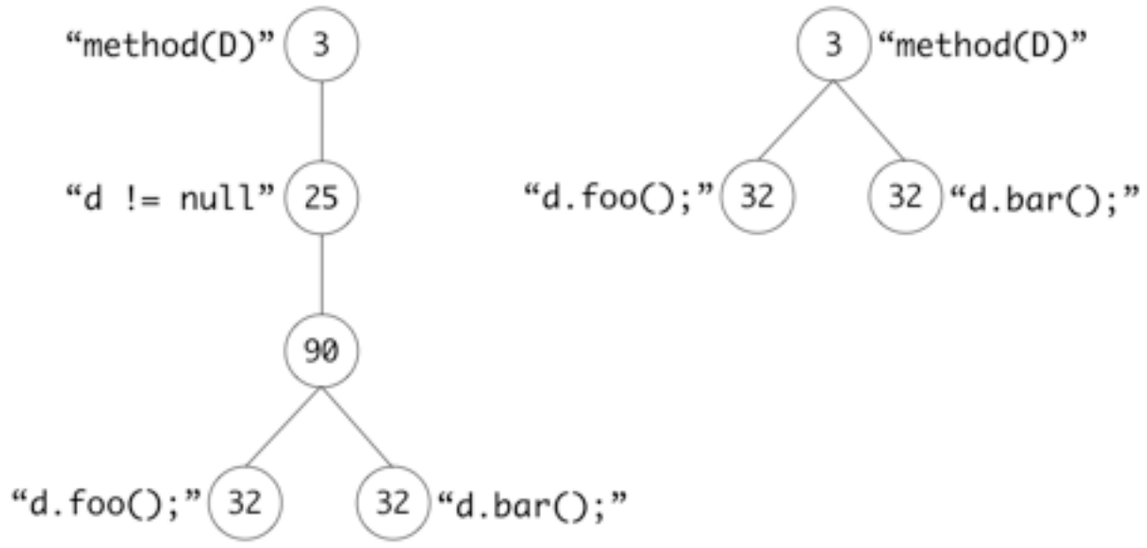
- Taxonomy classifies a single or a set of tree edit operations into a change type
  - body- or declaration-part change
  - name for the change according to the kind of the operations and involved tree-nodes, *e.g.*,
    - Statement Insert
    - Condition Expression Change
    - Method Renaming

Change Analysis

```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();

CVS log: “lines: +2 -4”
```

- 3 Body changes
  - 2 Statement parent changes
  - 1 Statement delete
  - Change significance

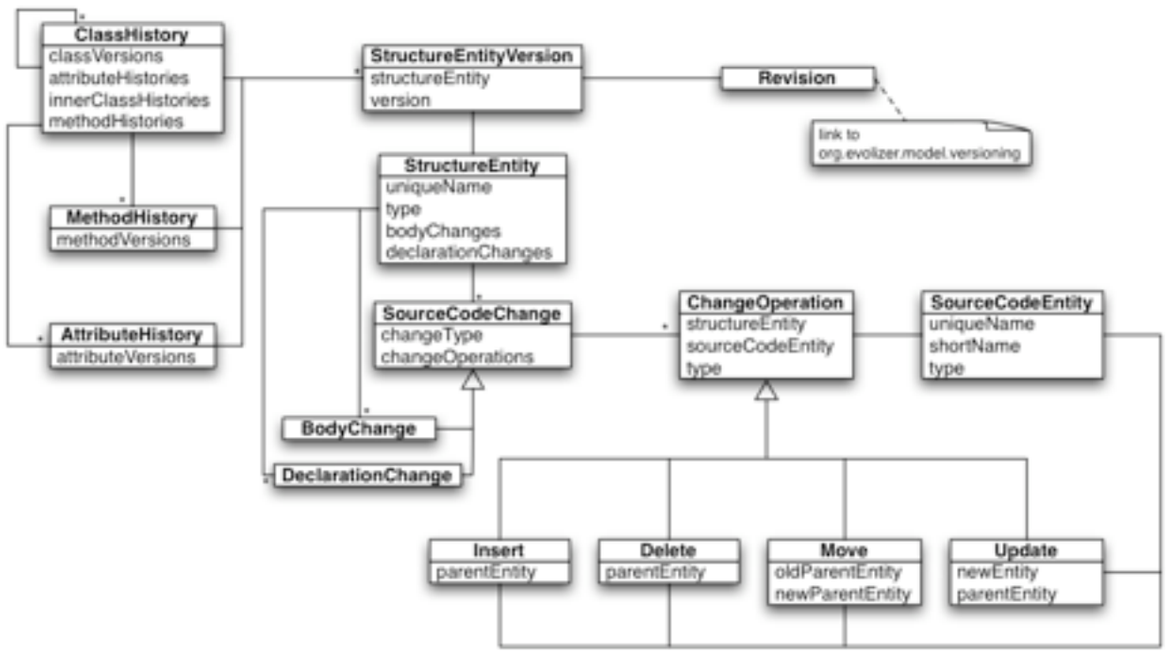
Building a Matching Set



Taxonomy of Source Code Changes

- Taxonomy classifies a single or a set of tree edit operations into a change type
  - body- or declaration-part change
  - name for the change according to the kind of the operations and involved tree-nodes, *e.g.*,
    - Statement Insert
    - Condition Expression Change
    - Method Renaming

ChangeDistiller Model

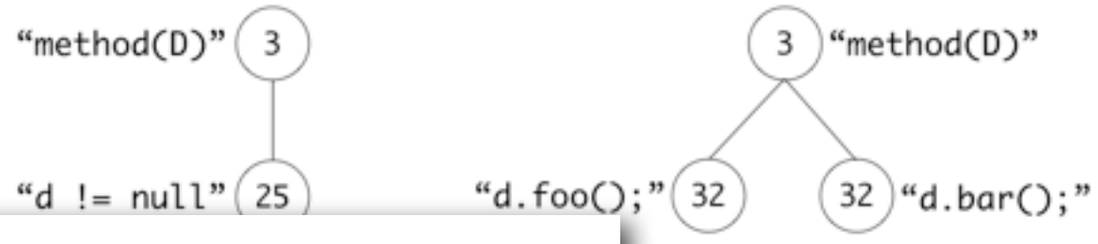


Change Analysis

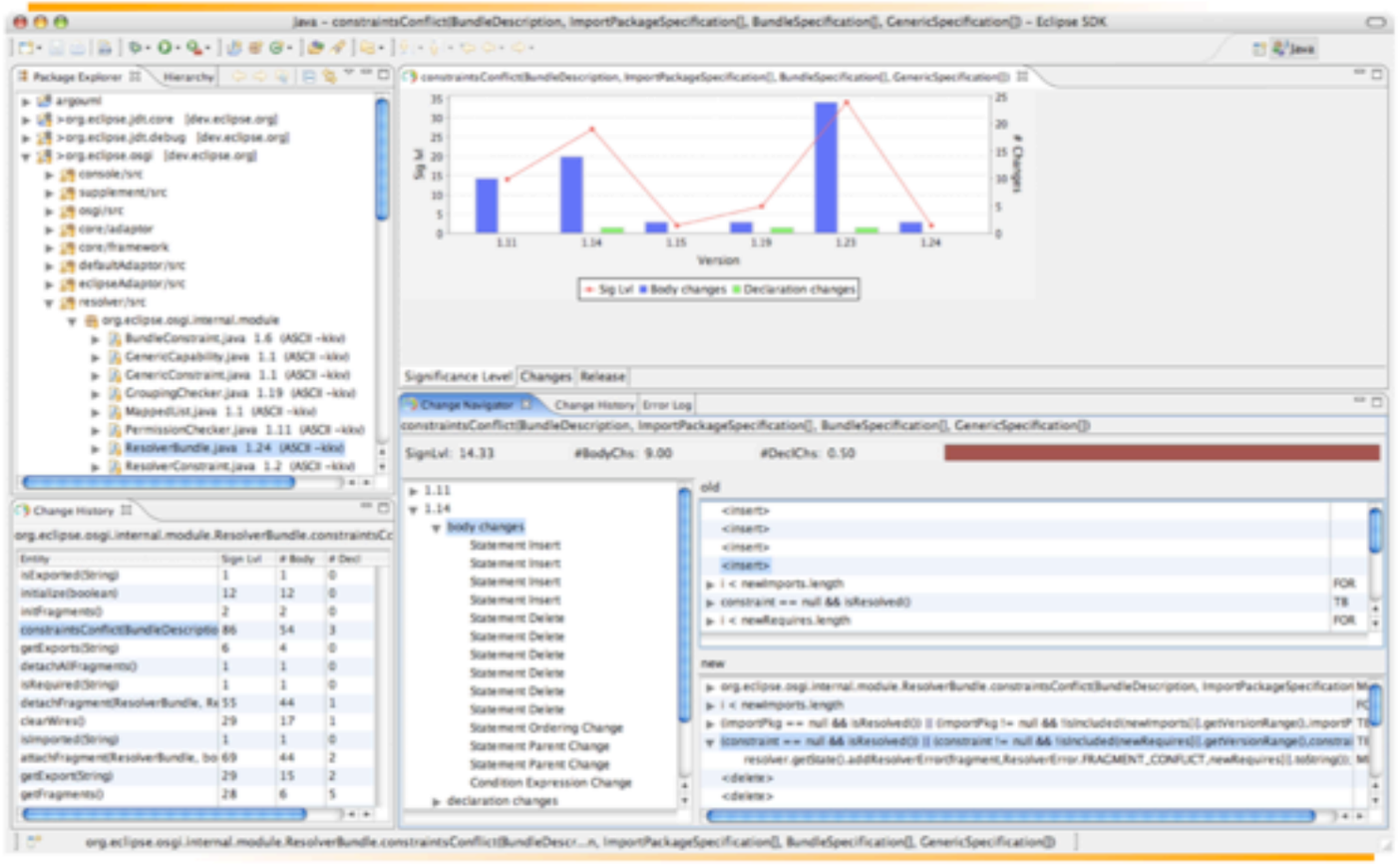
```
CVS diff

1967,1970c1964,1965
<     if (d != null) {
<         d.foo();
<         d.bar();
<     }
---
>     d.foo();
>     d.bar();

CVS log: “lines: +2 -
```

- 3 Body changes
  - 2 Statement parent changes

Building a Matching Set



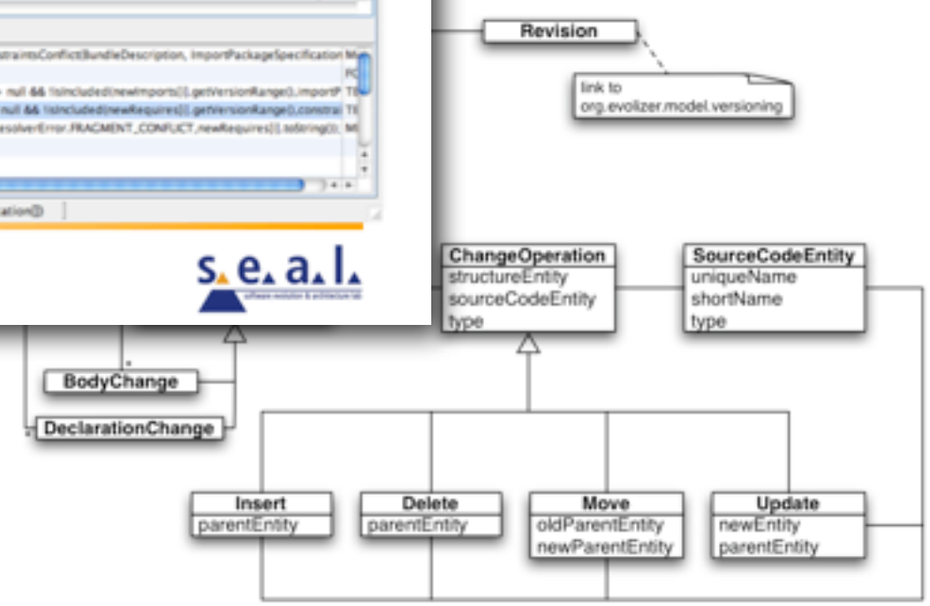
ChangeDistiller



Taxonomy of So

- Taxonomy classifie
  edit operations into
  - body- or declaratic
  - name for the chan
    operations and inv
    - Statement Insert
    - Condition Expression Change
    - Method Renaming

lel



Conclusions and Future Work

- Fine-grained source code changes according to tree edit operations
- Taxonomy of source code changes
- Assessing software quality criteria
- In future: further change significance analysis

Conclusions and Future Work

- Fine-grained source code changes according to tree edit operations
- Taxonomy of source code changes
- Assessing software quality criteria
- In future: further change significance analysis

Thank you

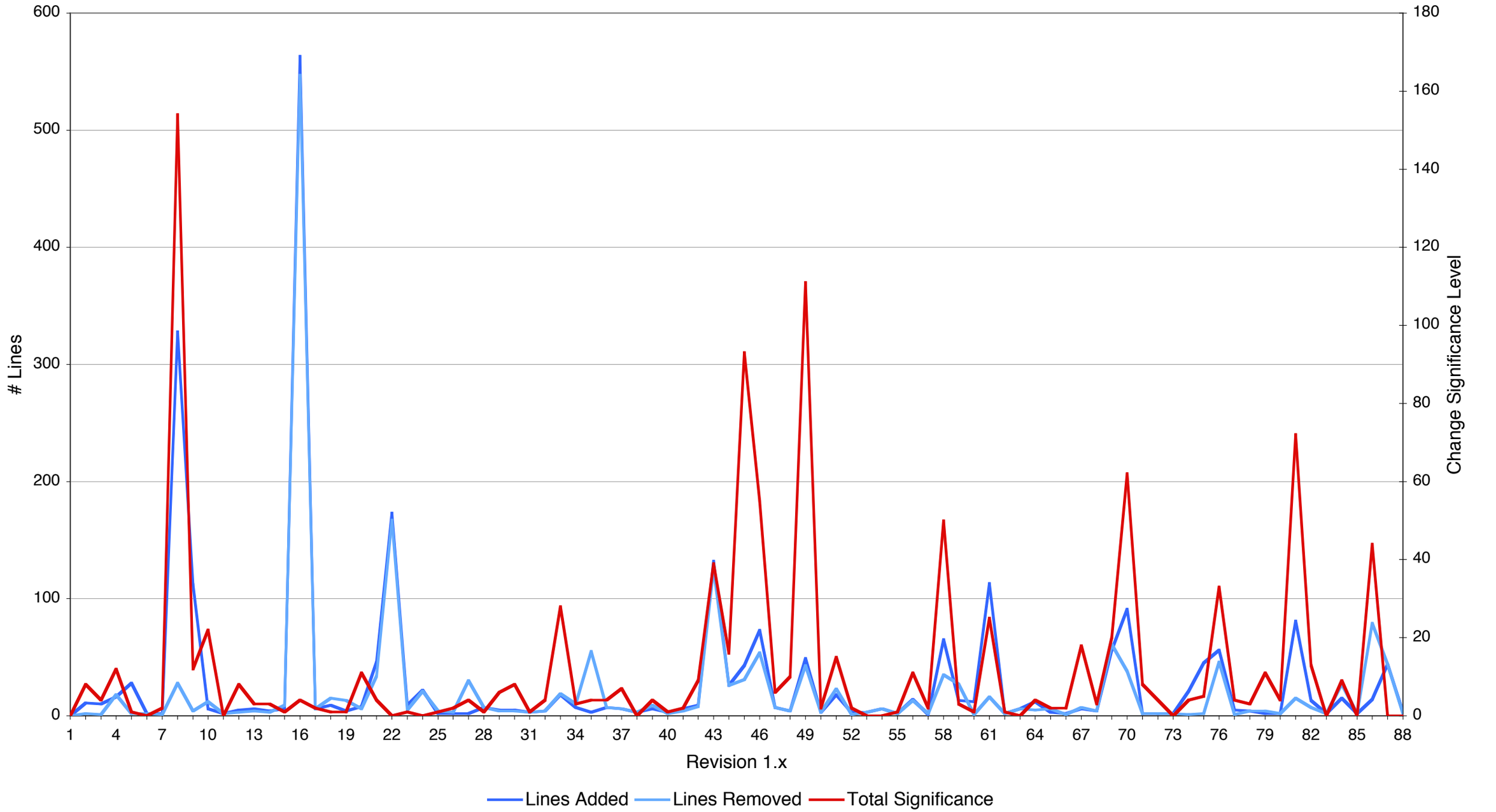
Implementation

- Eclipse Plugin **ChangeDistiller**
- ASTVisitor (JDT) to transform AST into intermediate tree
- Extracting tree edit operations using Chawathe's algorithm
- Classifying change types and storing in hibernate mapped database

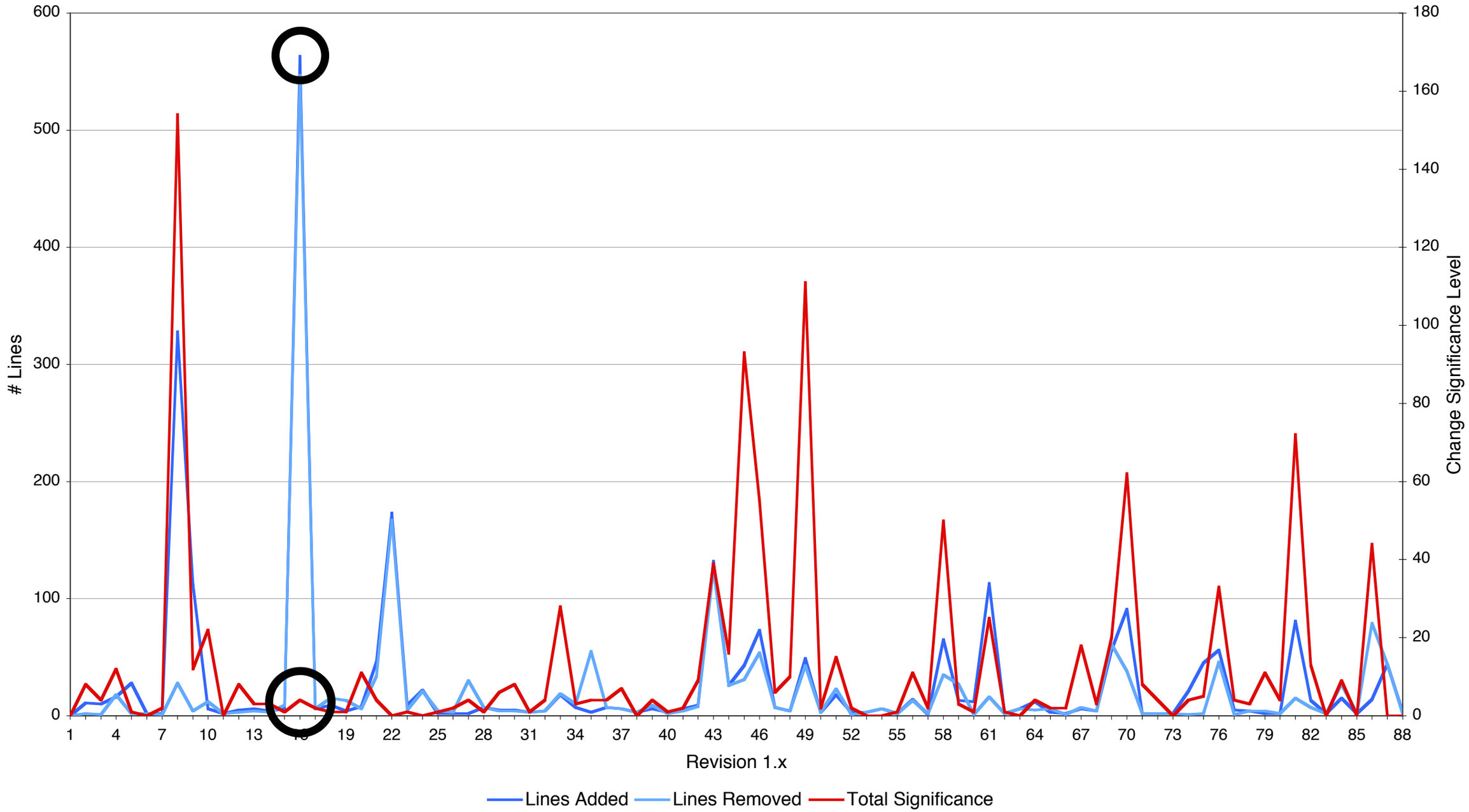
Case Study - ArgoUML

- Selected four classes having more than **80** revisions and a change coupling rate of **19**
- Research questions
  - To what extent are lines added/removed from CVS log indicators for the significance of the applied change?
  - Do the significance levels of change coupled files behave similarly?

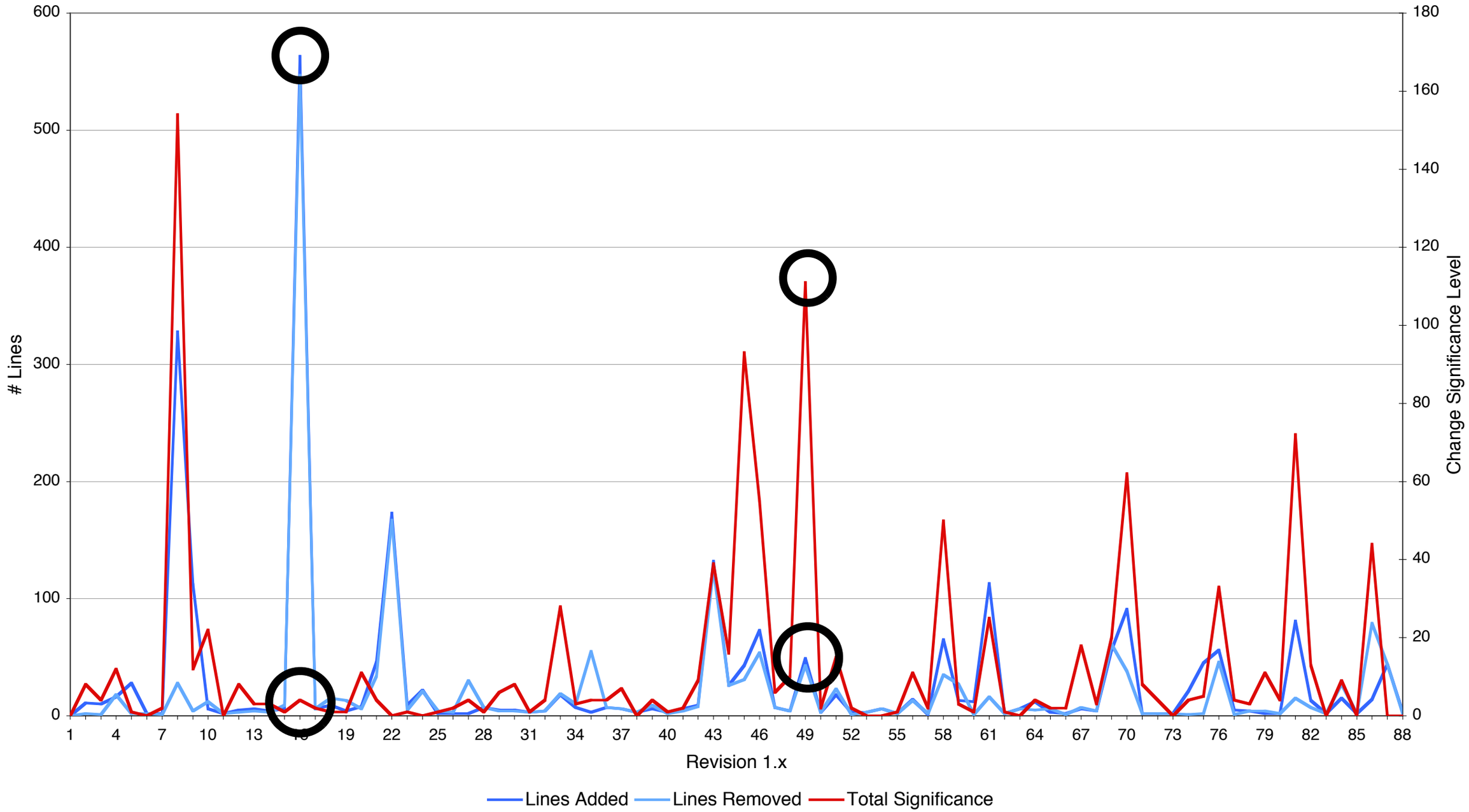
Change Significance Level



Change Significance Level



Change Significance Level



Change Significance Level

- Examples have shown that lines added/ removed are not indicators for the change significance
- Change significance level is more precise, *e.g.*, for representing change effort

Significant Change Couplings