抽象構文木内の階層移動に着目したGumTreeアルゴリズムの拡張

SIGSE　Mar 3, 2025

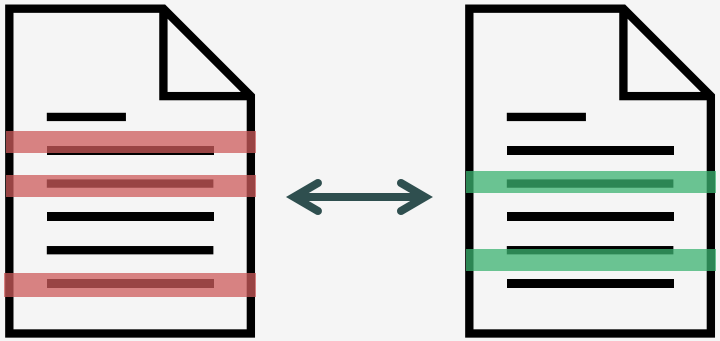
大阪大学大学院情報科学研究科
コンピュータサイエンス専攻
ソフトウェア工学講座

博士前期課程 2年　山本貴之

Department of Computer Science, Graduate School of Information Science and Technology, Osaka University.
Software Engineering Laboratory. - https://sel.ist.osaka-u.ac.jp

ソースコード差分

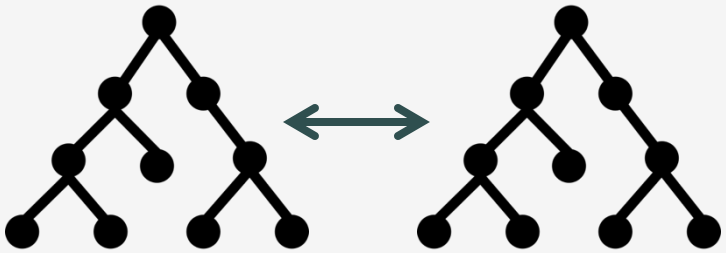
テキストベースの差分



変更前　　変更後

Unix diffコマンド
Git diffサブコマンド

抽象構文木（AST）を用いた差分

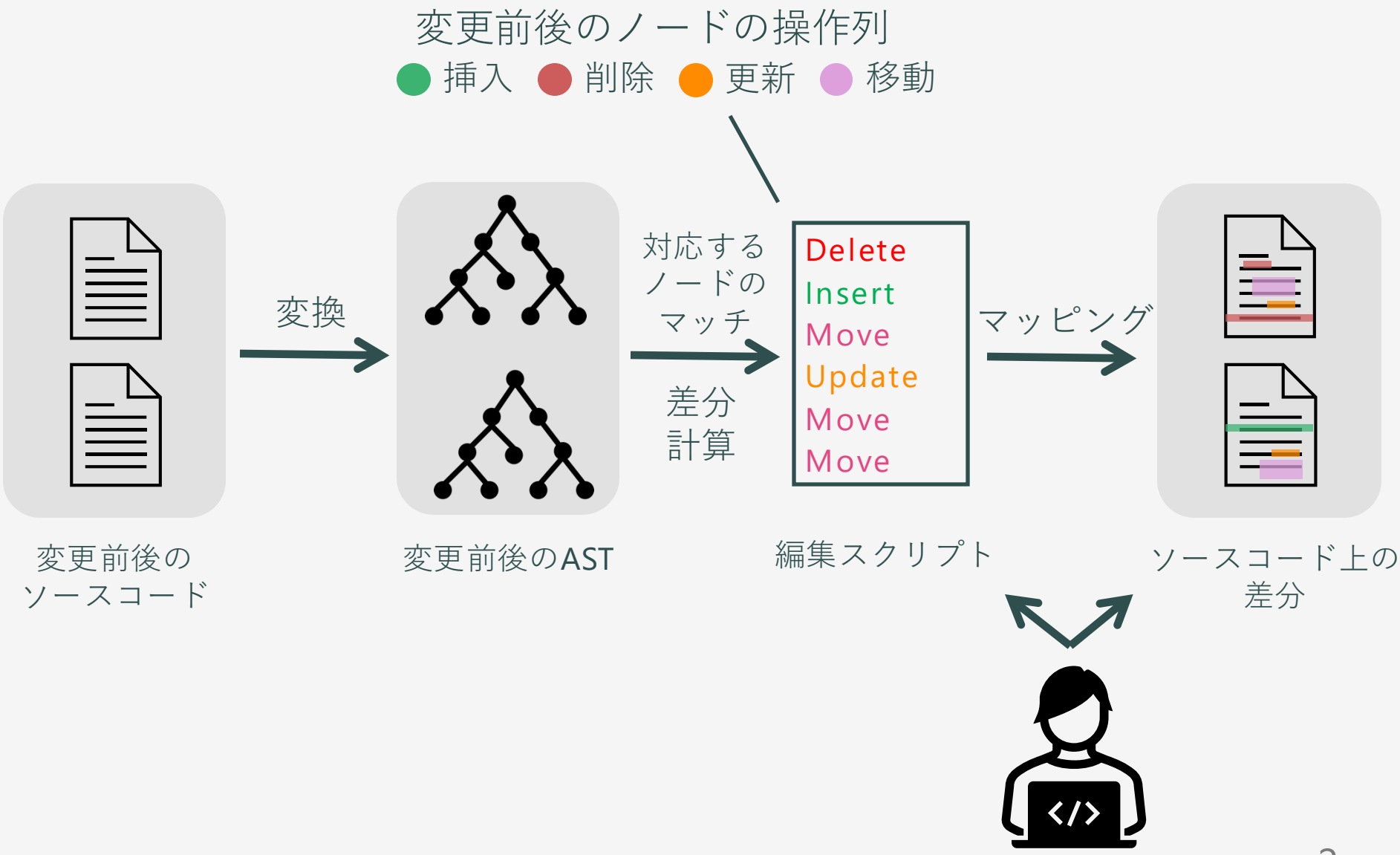


変更前　　変更後

GumTree[1][2]

[1] J. Falleri etc. “Finegrained and accurate source code differencing” in Proc. of the 29th ACM/IEEE ASE, 2014, pp. 313–324
[2] J. Falleri etc. “Fine-grained, accurate and scalable source differencing” in Proc. of the 46th IEEE/ACM ICSE. 2024, pp. 1-12

GumTreeの差分検出方法



GumTreeを用いたソースコード差分出力例

変更前

Calc.java

```
public class Calc {
  boolean calc(int i) {
    System.out.println(i);
    return i % 3 == 0;
  }
}
```

変更後

Calc.java

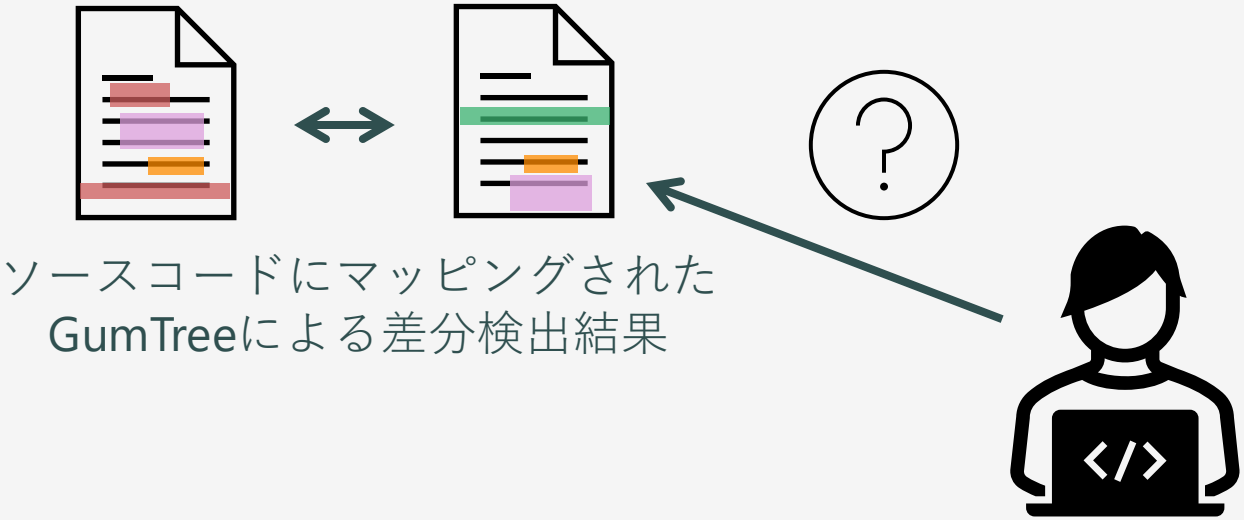
```
public class Calc {
  boolean divisible(int i) {
    if (i % 3 == 0) {
      return true;
    }
    return false;
  }
}
```

挿入 削除 更新 移動

単なるノードの操作列である編集スクリプトと比べ
ソースコード上の差分は視覚的に容易に理解可能

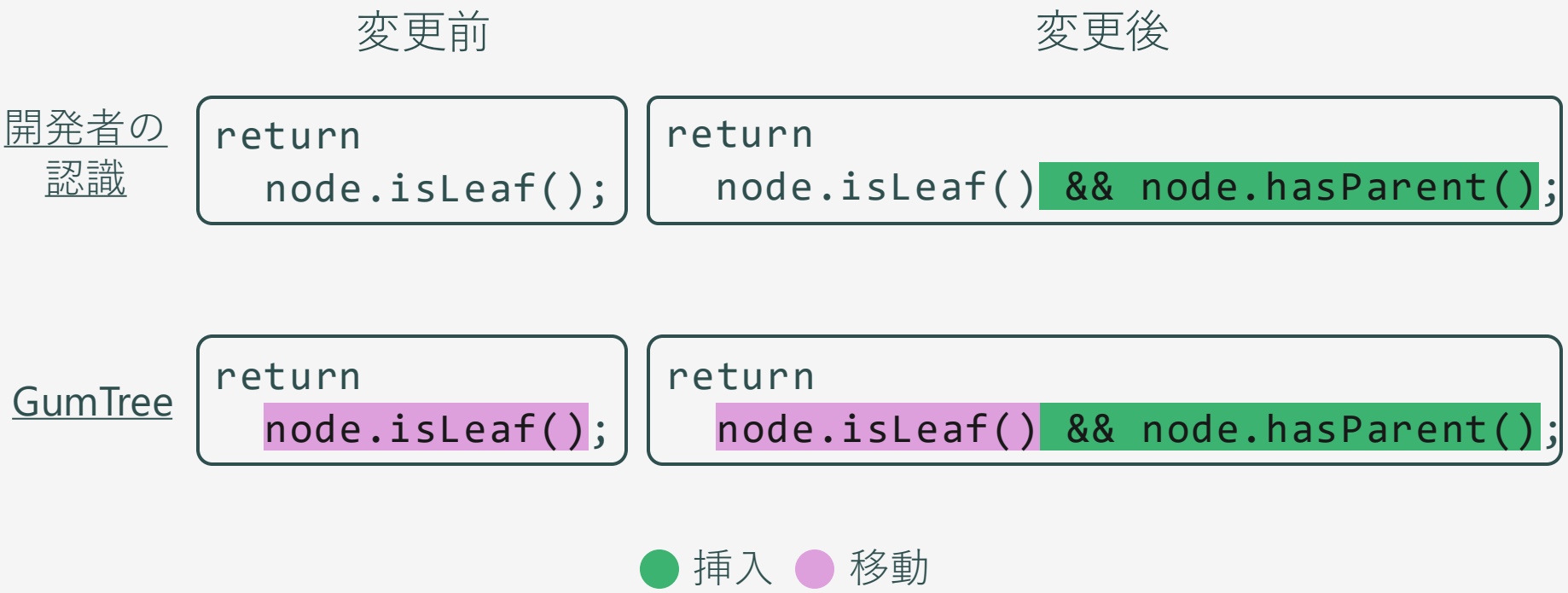
GumTreeの問題点

GumTreeの差分検出結果と
開発者がソースコード上で認識する差分情報
に乖離がある



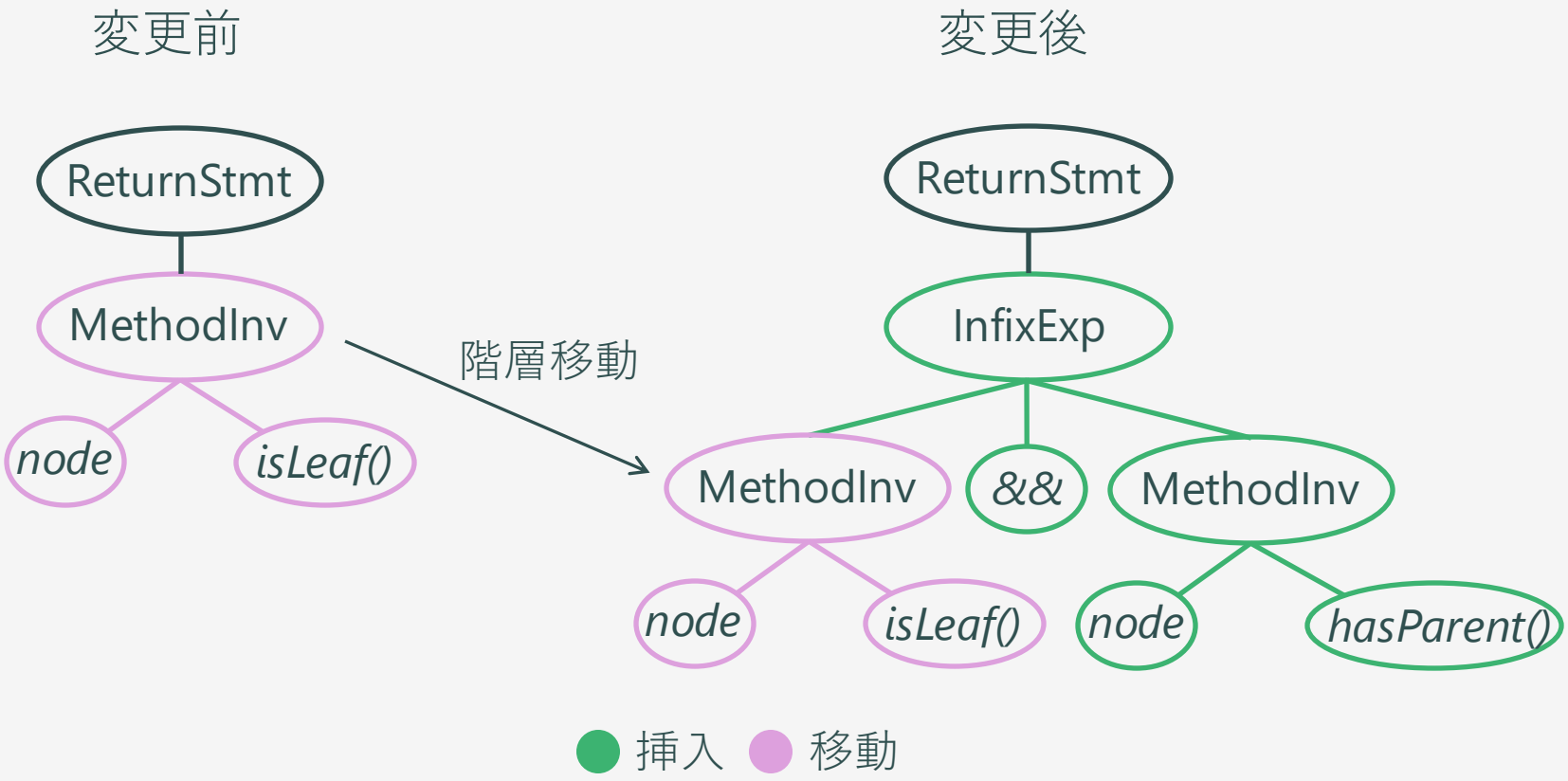
ソースコードにマッピングされた
GumTreeによる差分検出結果

差分認識が乖離する例



開発者はなぜ**移動**と判定されているのか分からない

GumTreeが移動を検出した原因



ソースコード上では変化しないコード片がASTでは移動と判定される

本研究の目的・提案

目的

より**視覚的に**開発者が理解しやすい差分検出を可能にする

提案

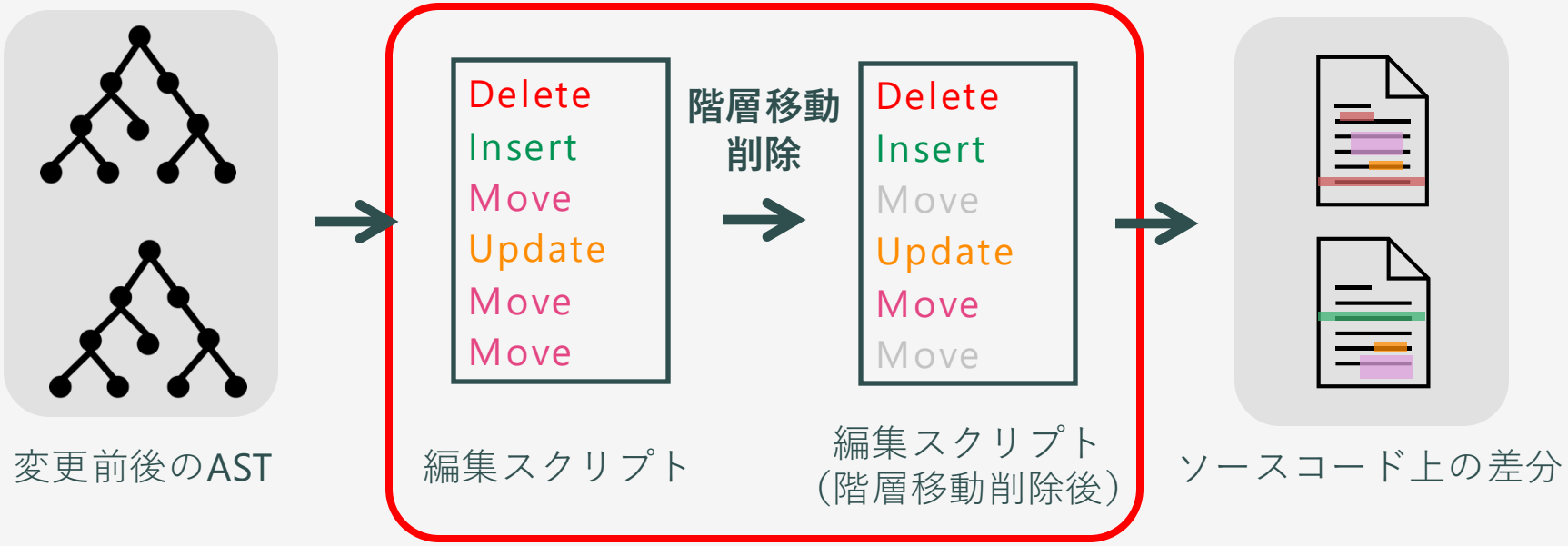
GumTreeを拡張し，差分理解を妨げる階層移動を
検出・削除する手法を提案

プログラミング言語：Java
AST生成ライブラリ：Eclipse JDT Core*（JDT）

* Eclipse JDT Core: GumTreeがデフォルトで使用しているAST生成ライブラリ

階層移動の検出手法

GumTreeを拡張し，新たに階層移動判定を追加する



階層移動判定アルゴリズム

階層移動の検出手法

GumTreeを拡張し，新たに階層移動判定を追加する

検出アルゴリズムを提案する前に！！！
- **階層移動の定義**
- **階層移動が起こりうるノードの調査**

階層移動判定アルゴリズム

検出・削除対象となる階層移動の定義

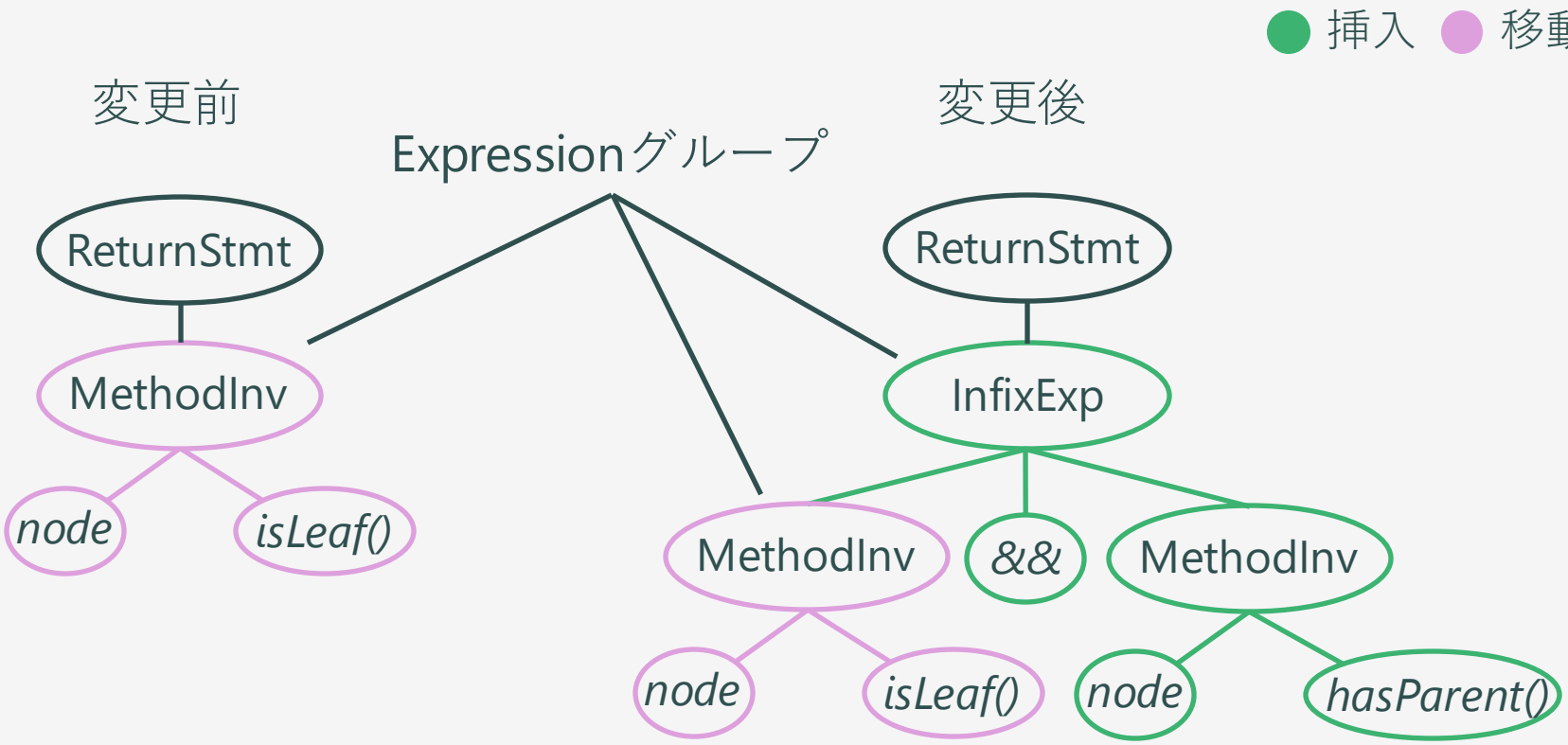
条件１：同種類のノード間で完結する移動

条件２：Statementよりも粒度が細かいノードで生じた移動
（if文のelseブロックにおける移動は例外）

条件３：横移動を含まない移動

※ 種類はJDTの情報を基に分類

検出・削除対象の階層移動例
（条件1：同種類のノード間で完結する移動）

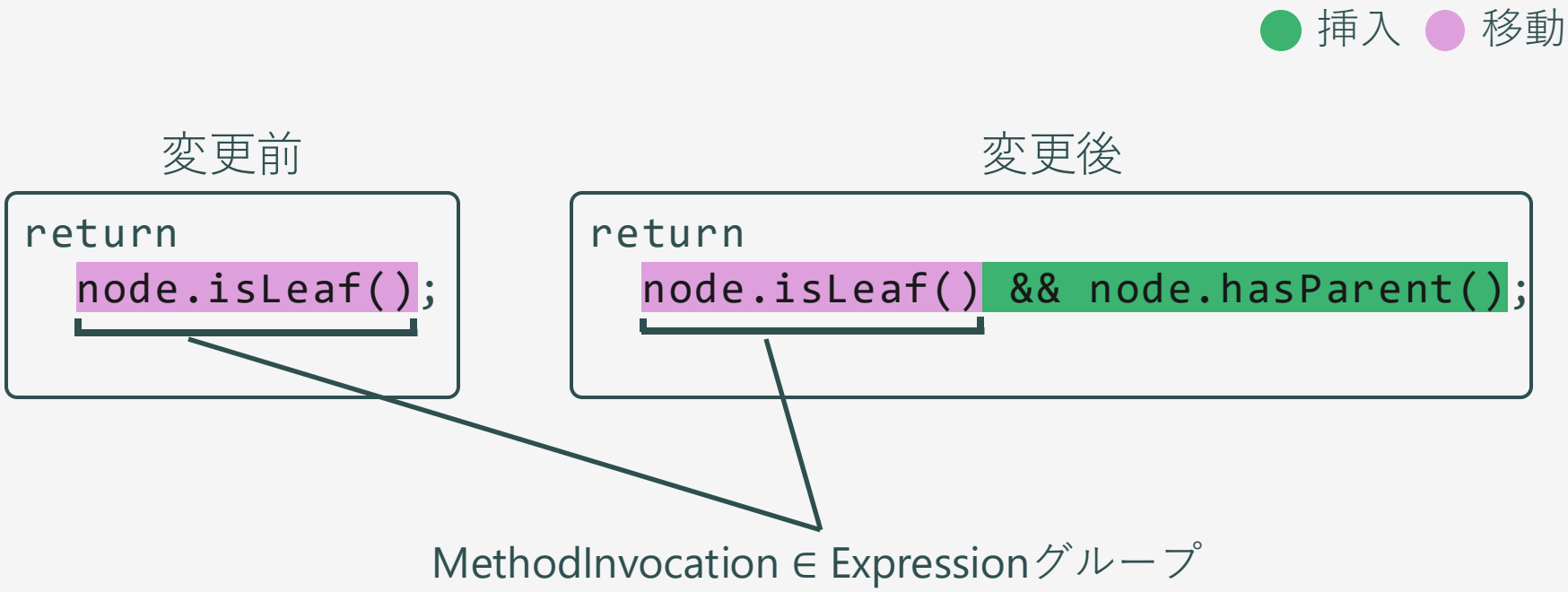



```
return
  node.isLeaf();
```

```
return
  node.isLeaf() && node.hasParent();
```

条件1：同種類のノード間で完結する

検出・削除対象の階層移動例
（条件2：Statementよりも粒度が細かい移動）



条件2：Statementよりも粒度が細かいノードが移動

検出・削除対象の階層移動例
（条件3：横移動を含まない移動）

● 挿入 ● 移動

変更前

```
return
  node.isLeaf();
```

変更後

```
return
  node.isLeaf() && node.hasParent();
```

条件3：横移動を含まない

検出・削除対象の階層移動例

● 挿入 ● 移動

変更前

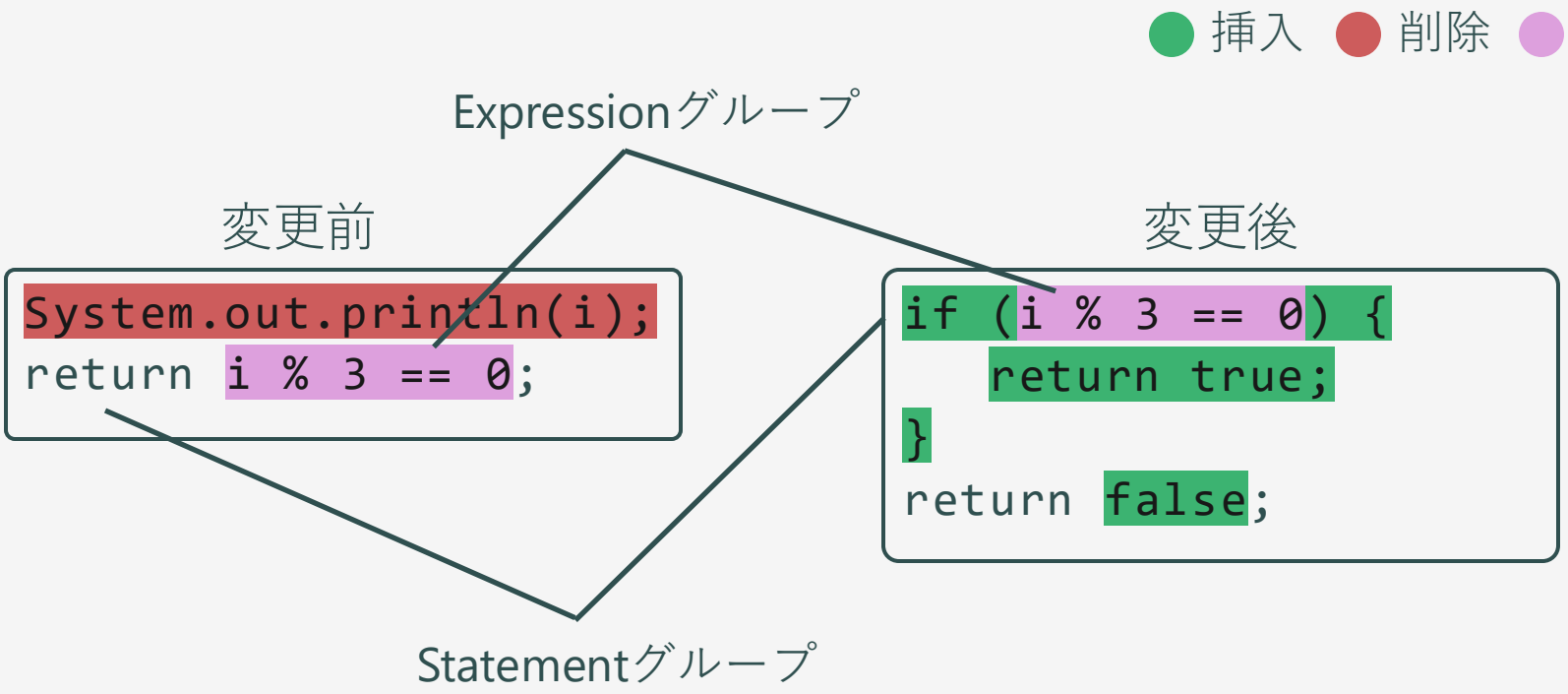
```
return
  node.isLeaf();
```

変更後

```
return
  node.isLeaf() && node.hasParent();
```

3つの条件を満たす！
階層移動であり移動情報を削除する必要あり

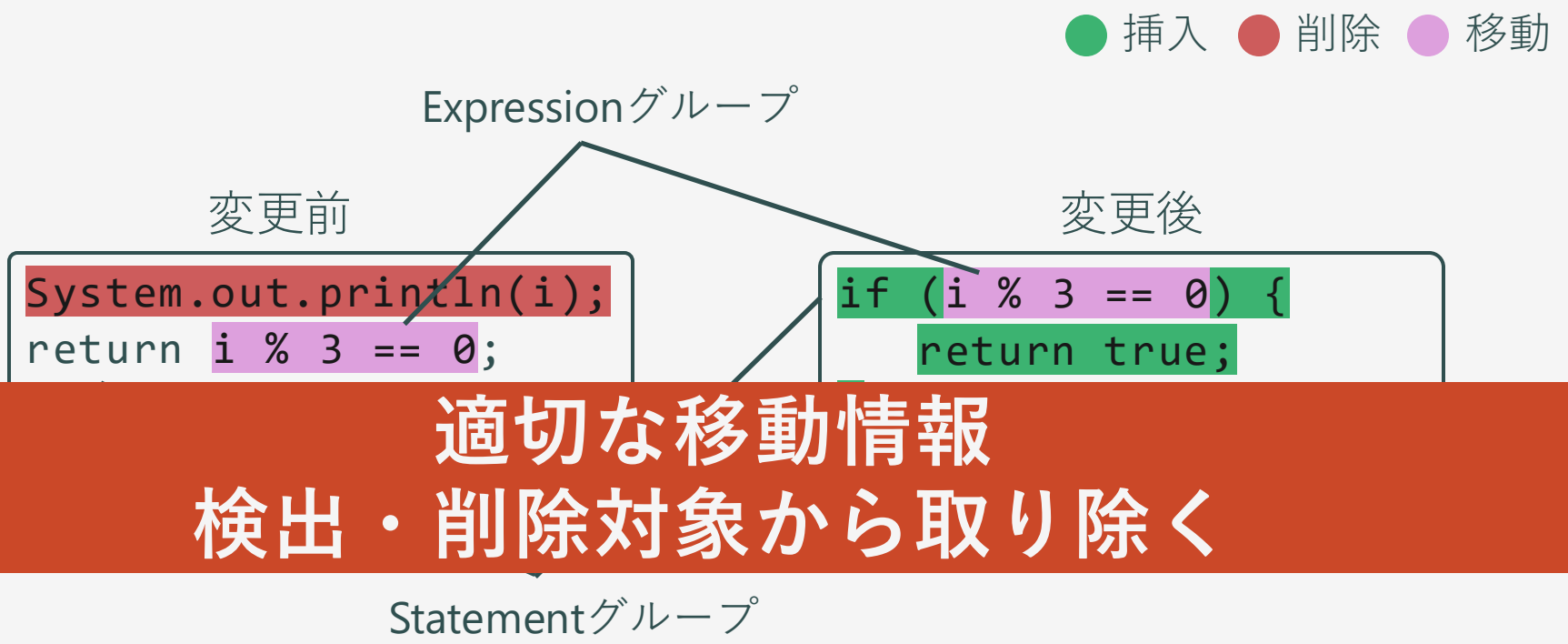
異種類のノードを跨ぐ移動例
（条件1：同種類のノード間で完結する移動）



Expressionが異なるStatementへ移動（return文→if文）

異種類のノードを跨ぐ移動は
ソースコード上でも明らかな移動を示す

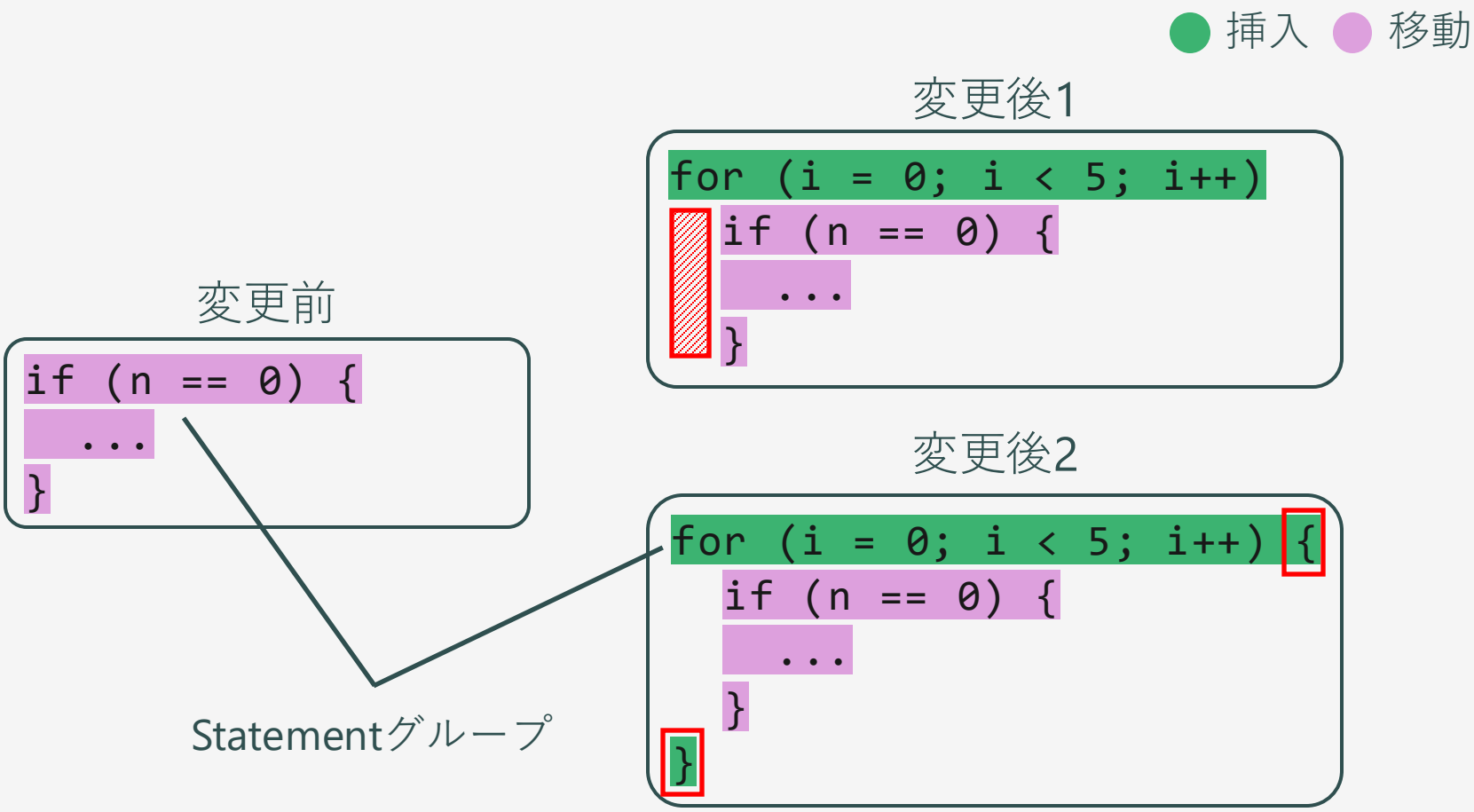
異種類のノードを跨ぐ移動例
（条件1：同種類のノード間で完結する移動）



Expressionが異なるStatementへ移動（return文→if文）

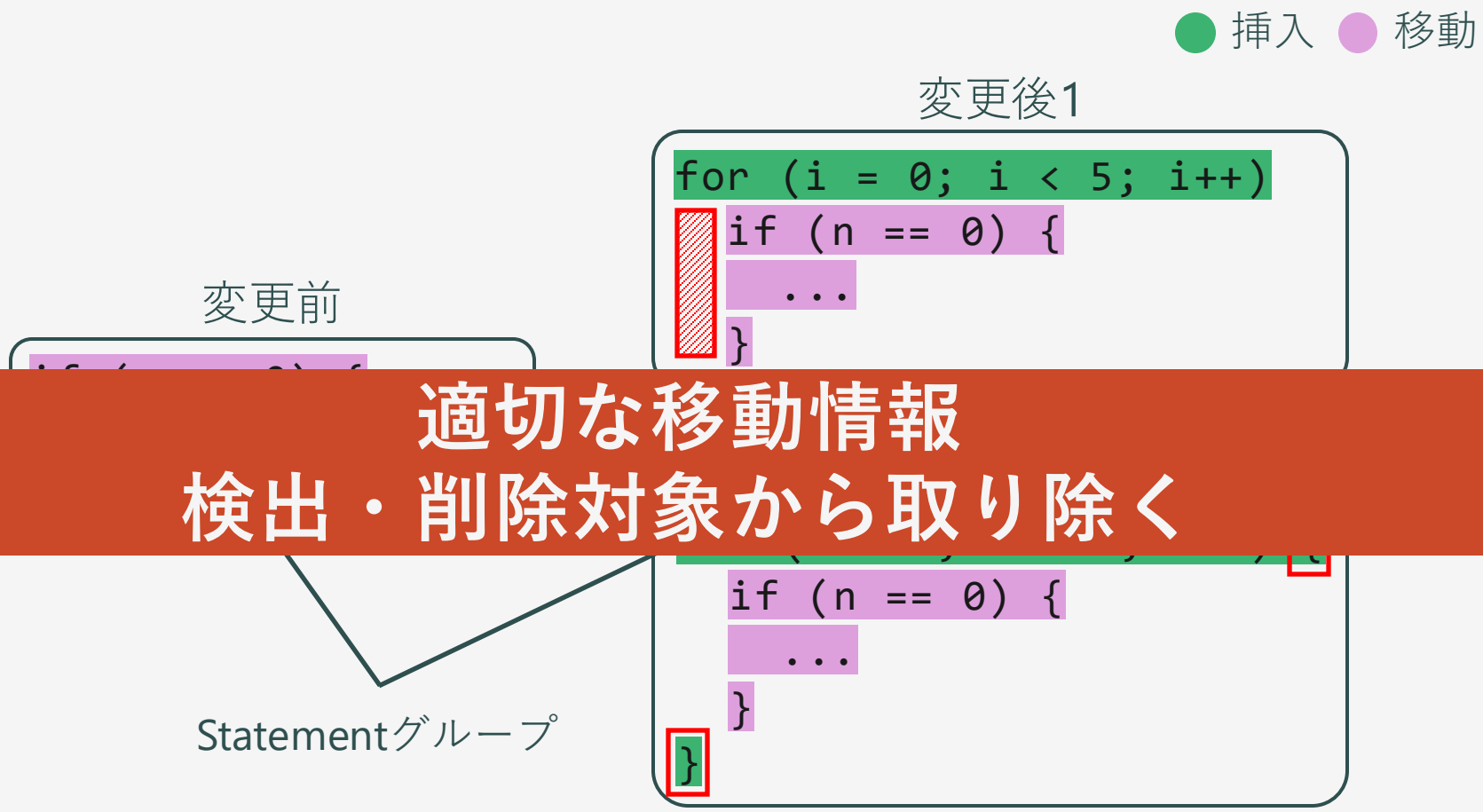
異種類のノードを跨ぐ移動は
ソースコード上でも明らかな移動を示す

Statement以上に粒度が粗い移動例
（条件2：Statementよりも粒度が細かい移動）



インデントや中括弧で**ソースコード上でも階層構造を表現**
ASTでの階層構造と開発者の認識の間に乖離がない

Statement以上に粒度が粗い移動例
（条件2：Statementよりも粒度が細かい移動）



インデントや中括弧で**ソースコード上でも階層構造を表現**
ASTでの階層構造と開発者の認識の間に乖離がない

横移動を含む移動例
（条件3：横移動を含まない移動）

変更前

```
return
  nodes[i].getName();
```

変更後

```
return
  getName((Block) nodes[i]);
```

nodes[i]がgetNameを境に横移動している

明らかに移動を認識できる

横移動を含む移動例
（条件3：横移動を含まない移動）

変更前

```
return
  nodes[i].getName();
```

変更後

```
return
  getName((Block) nodes[i]);
```

適切な移動情報
検出・削除対象から取り除く

nodes[i]がgetNameを境に横移動している

明らかに移動を認識できる

階層移動が起こりうるノードの調査

目的

ASTの構造的に階層移動が起こりうるノードを調査し
アルゴリズムの適用範囲を予め限定する

JDTにおける
ASTの構造から
事前に調査可能

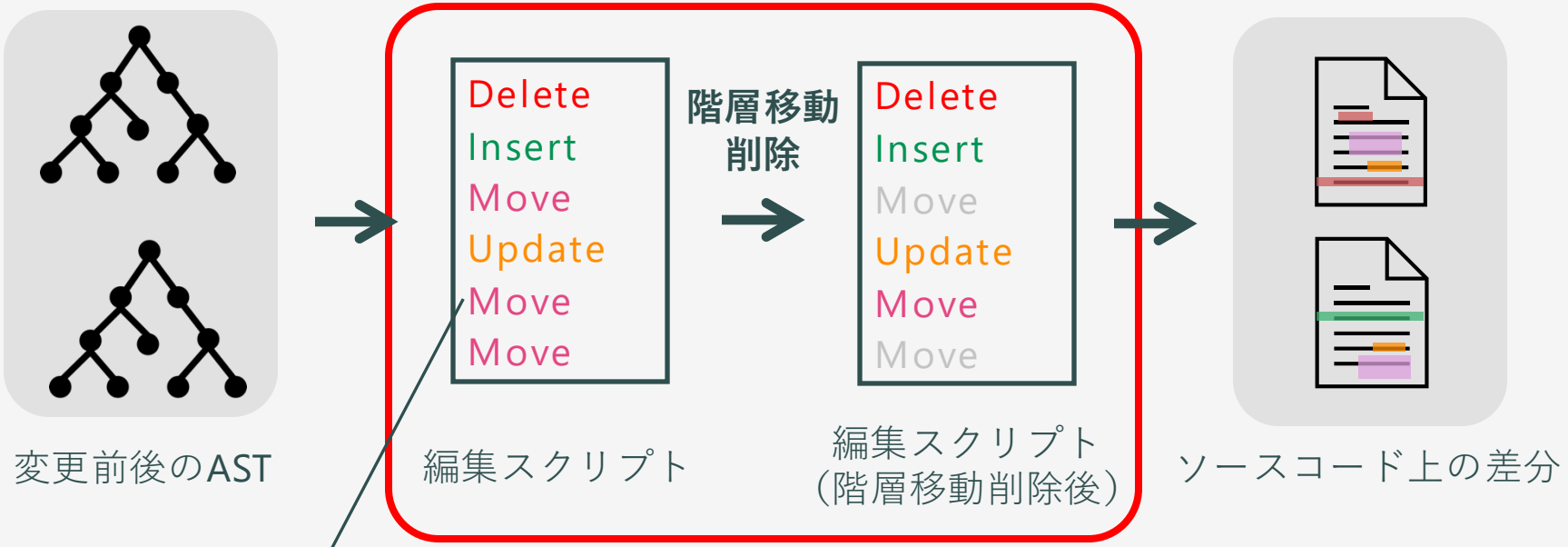
条件１：同種類のノード間で完結する移動
条件２：Statementよりも粒度の細かいノードで生じた移動
（if文のelseブロックにおける移動は例外）
条件３：横移動を含まない移動

階層移動が発生する可能性のあるノードの種類一覧

Expression Name Pattern Type if文のelse構造

これら**5種類**に分類されるノードの移動のみ検査する

階層移動判定アルゴリズム



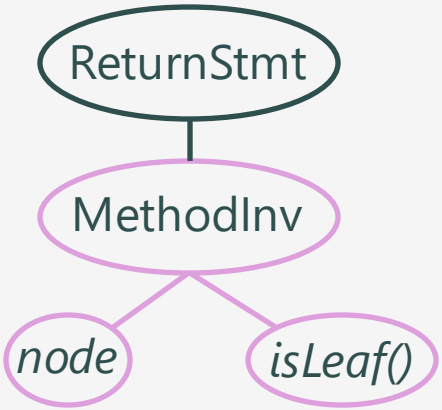
階層移動判定アルゴリズム

STEP1：調査した5種類に含まれる（条件2）
STEP2：同種類のノード内で完結する（条件1）
STEP3：横移動を含まない（条件3）

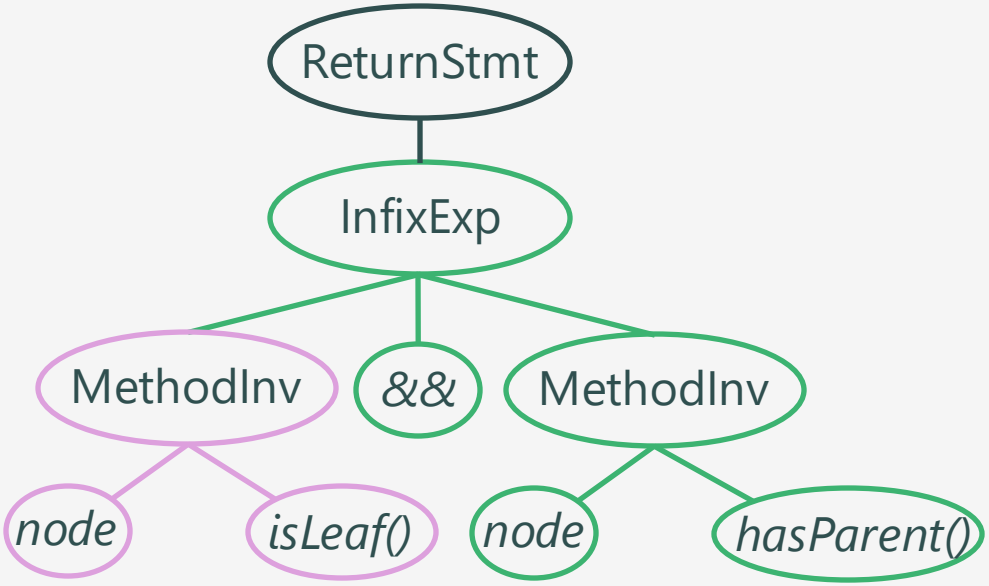
階層移動判定アルゴリズム適用例（再掲）

挿入 移動

変更前



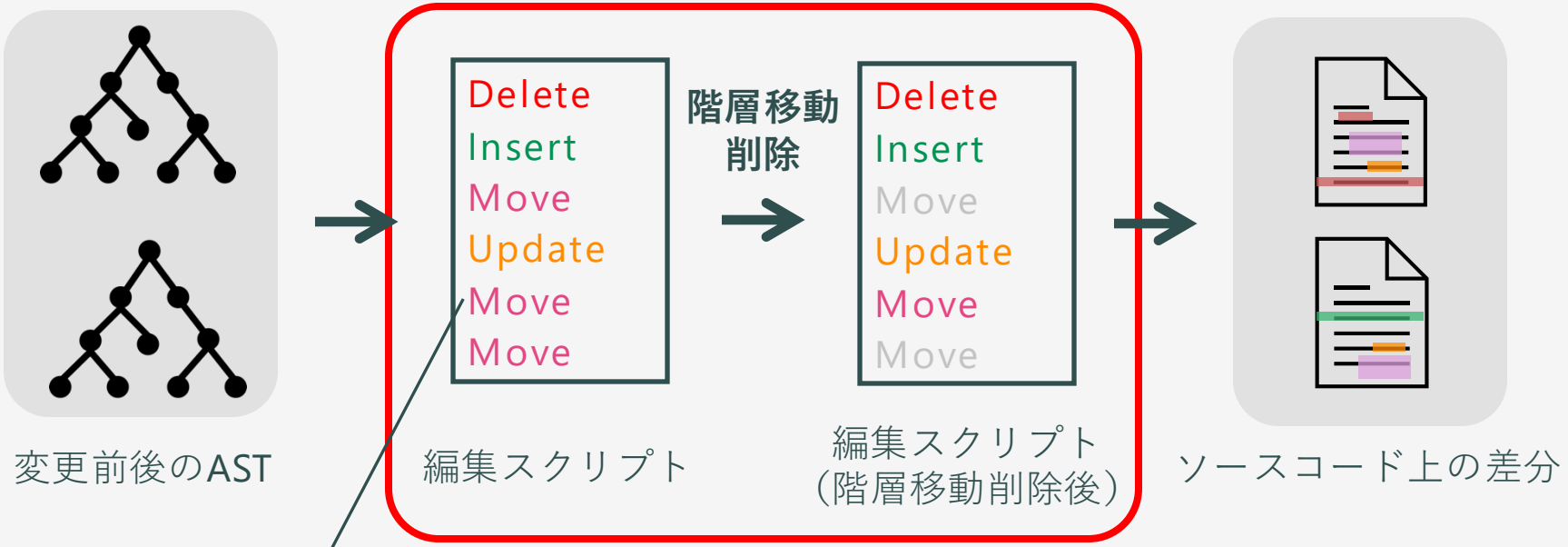
変更後




```
return
  node.isLeaf();
```

```
return
  node.isLeaf() && node.hasParent();
```

階層移動判定アルゴリズム



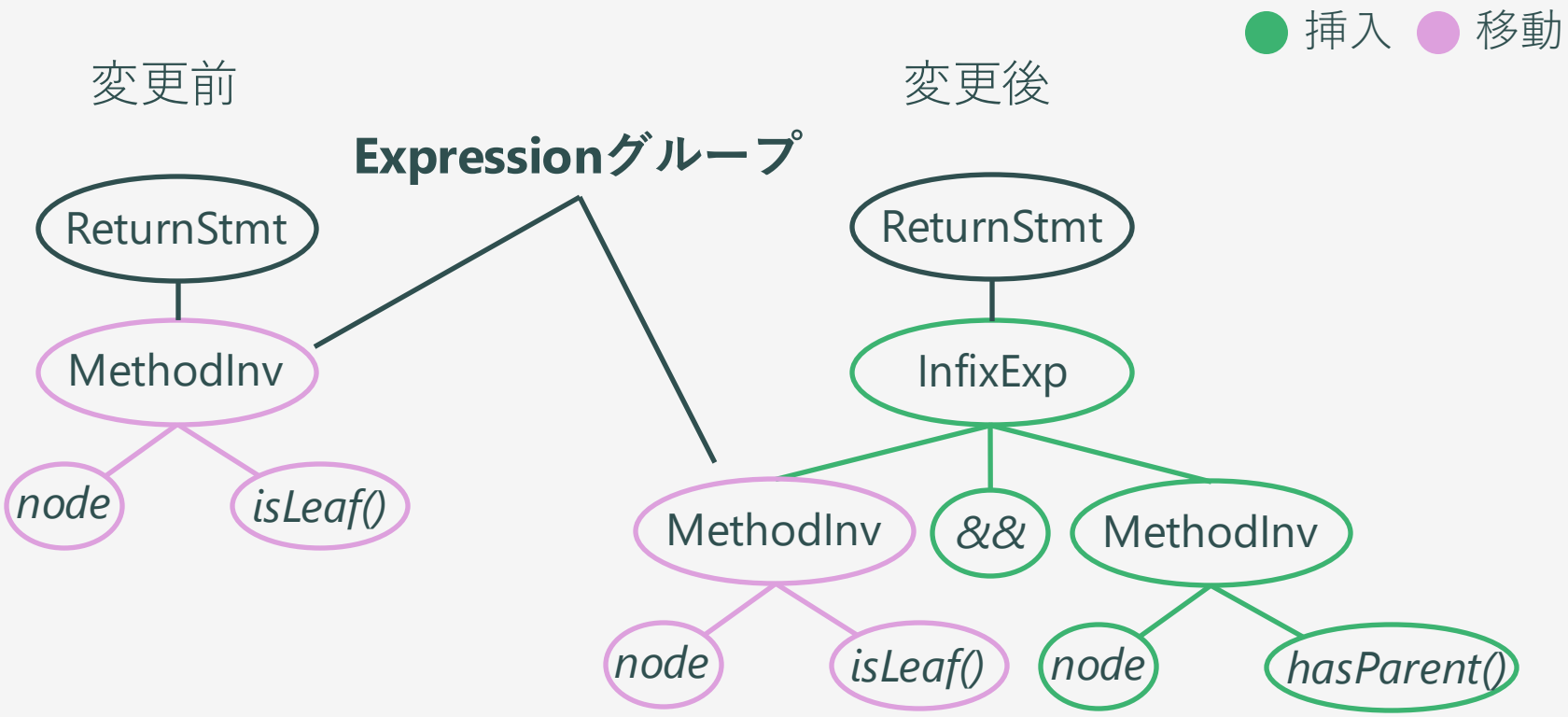
階層移動判定アルゴリズム

STEP1：調査した5種類に含まれる（条件2）

STEP2：同種類のノード内で完結する（条件1）

STEP3：横移動を含まない（条件3）

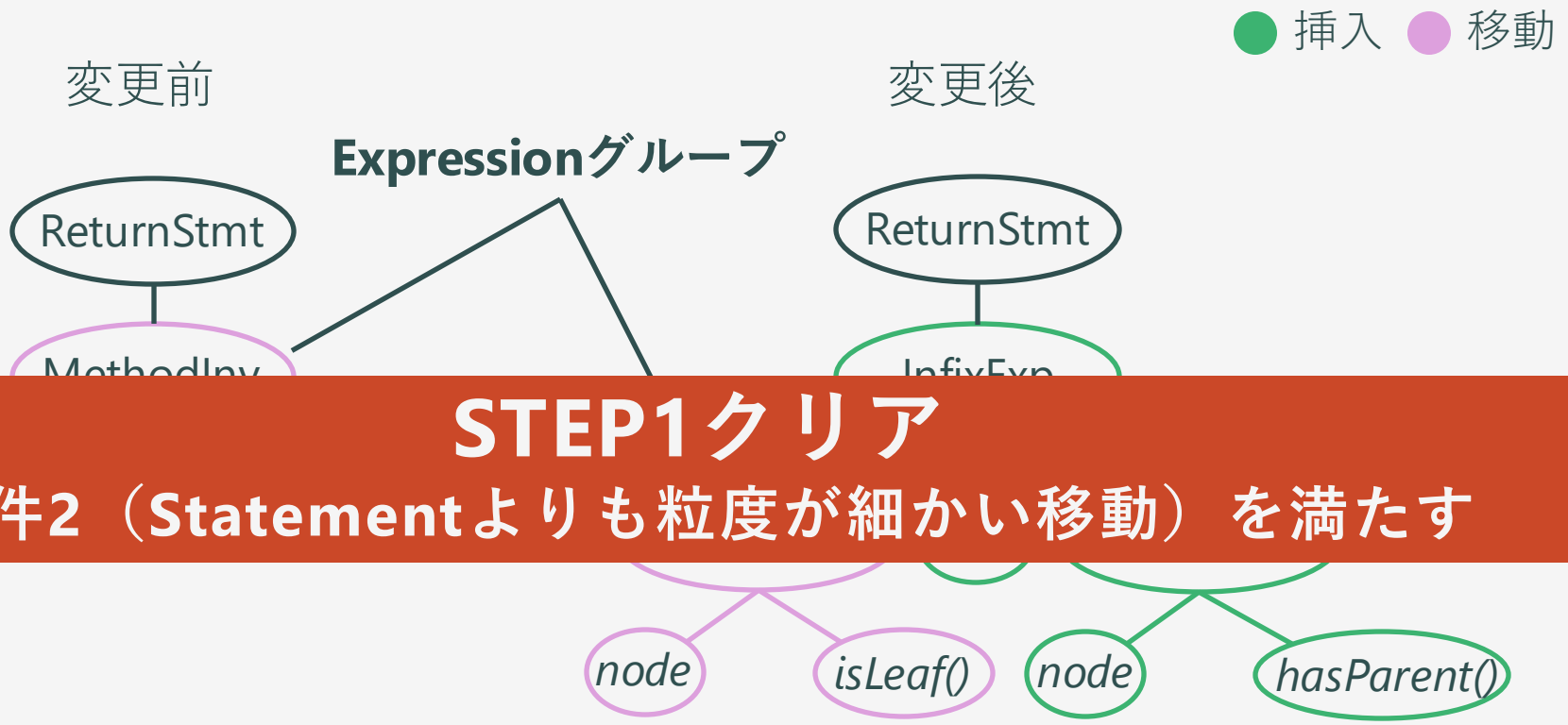
STEP1: 調査した5種類に含まれる（条件2）



調査結果

Expression Name Pattern Type if文のelse構造

STEP1: 調査した5種類に含まれる（条件2）



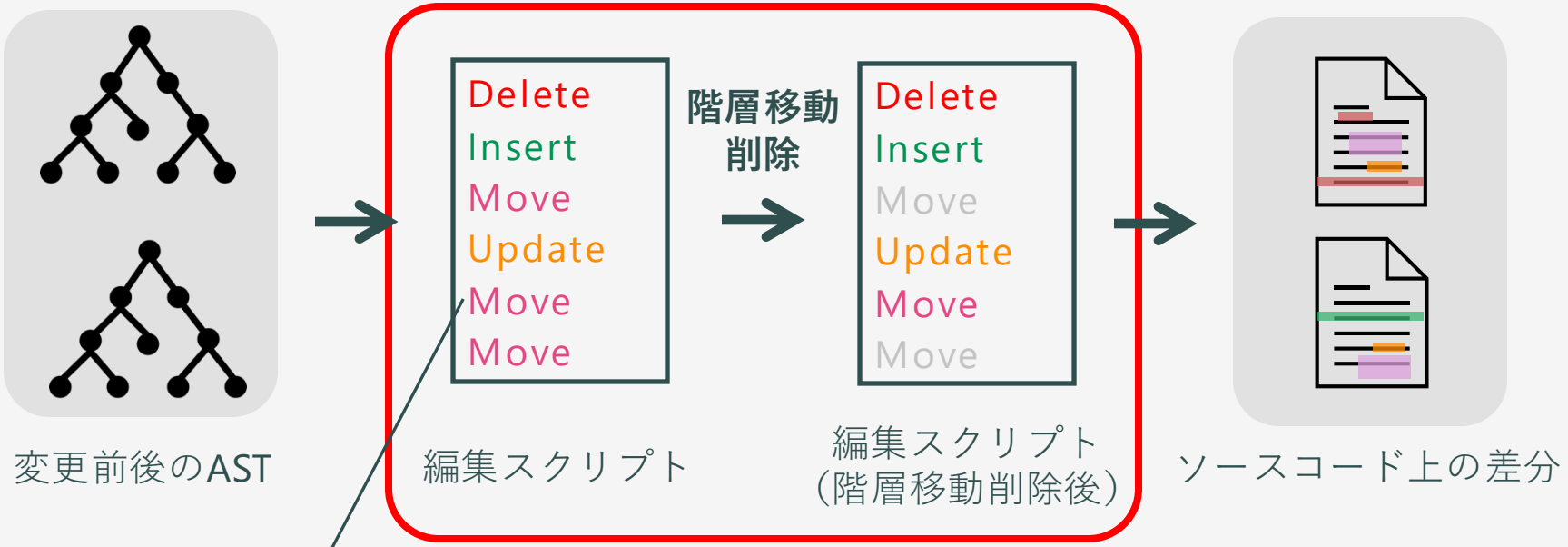
STEP1クリア

条件2（Statementよりも粒度が細かい移動）を満たす

調査結果

Expression Name Pattern Type if文のelse構造

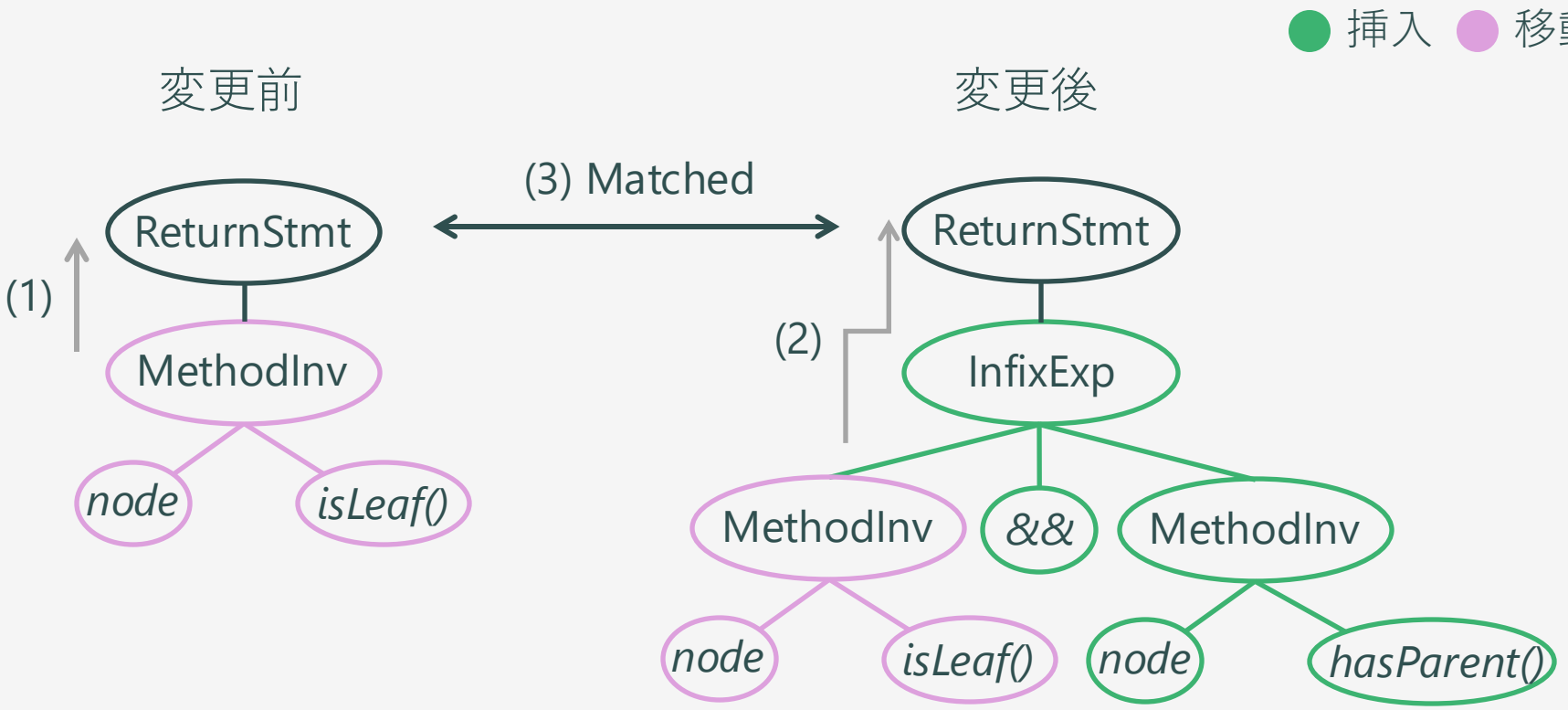
階層移動判定アルゴリズム



階層移動判定アルゴリズム

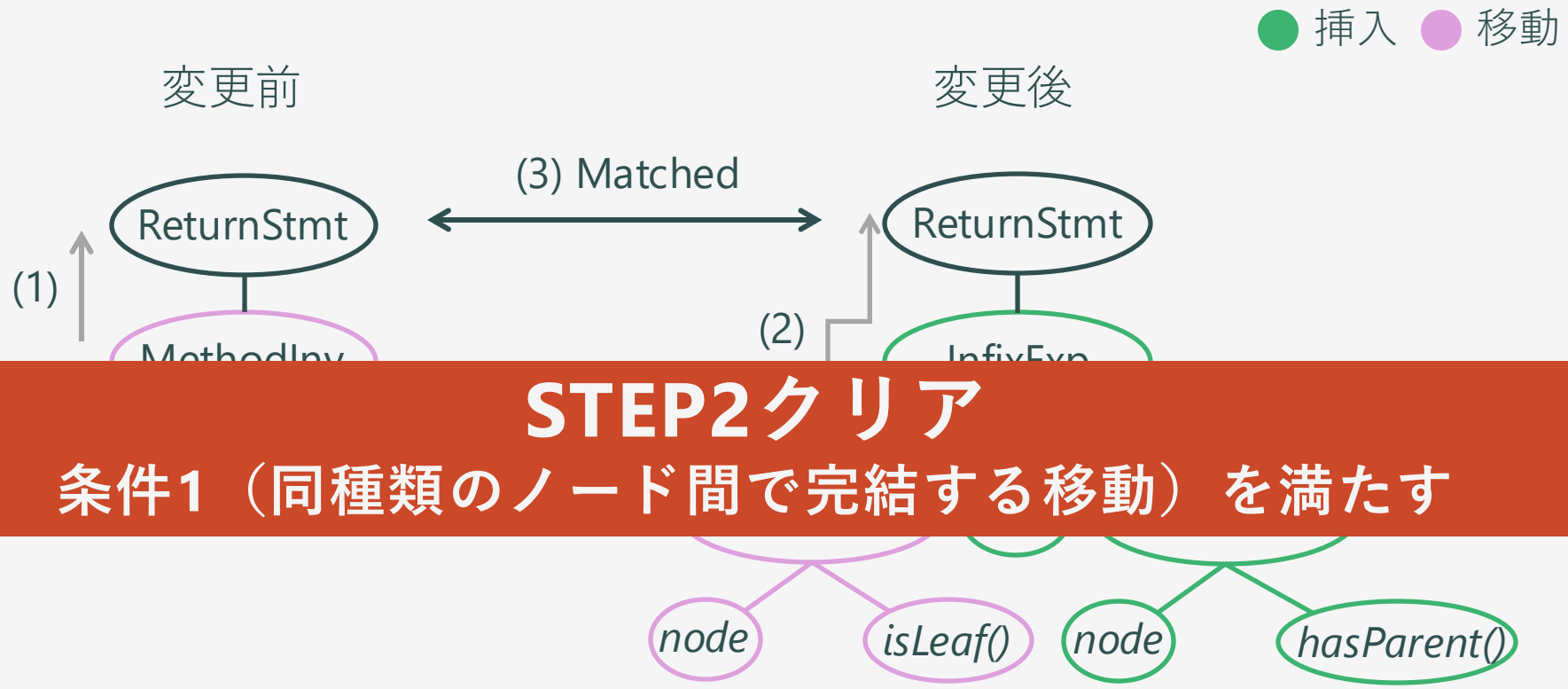
STEP1：調査した5種類に含まれる（条件2）
STEP2：同種類のノード内で完結する（条件1）
STEP3：横移動を含まない（条件3）

STEP2: 同種類のノード間で完結する（条件1）



根ノードに向けて異種類のノードに到達するまで遡る
到達したノードがマッチしていないと条件1に反する

STEP2: 同種類のノード間で完結する（条件1）

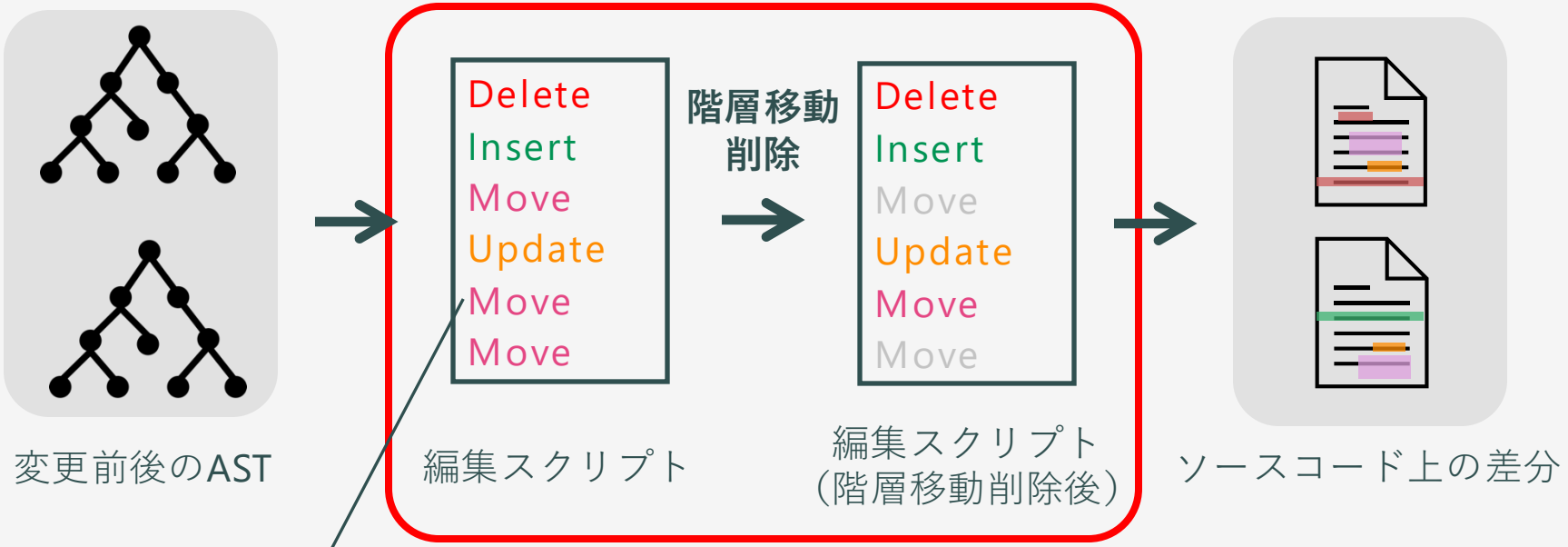


STEP2クリア

条件1（同種類のノード間で完結する移動）を満たす

マッチしない場合は異種類のノードを跨いで移動したことになる

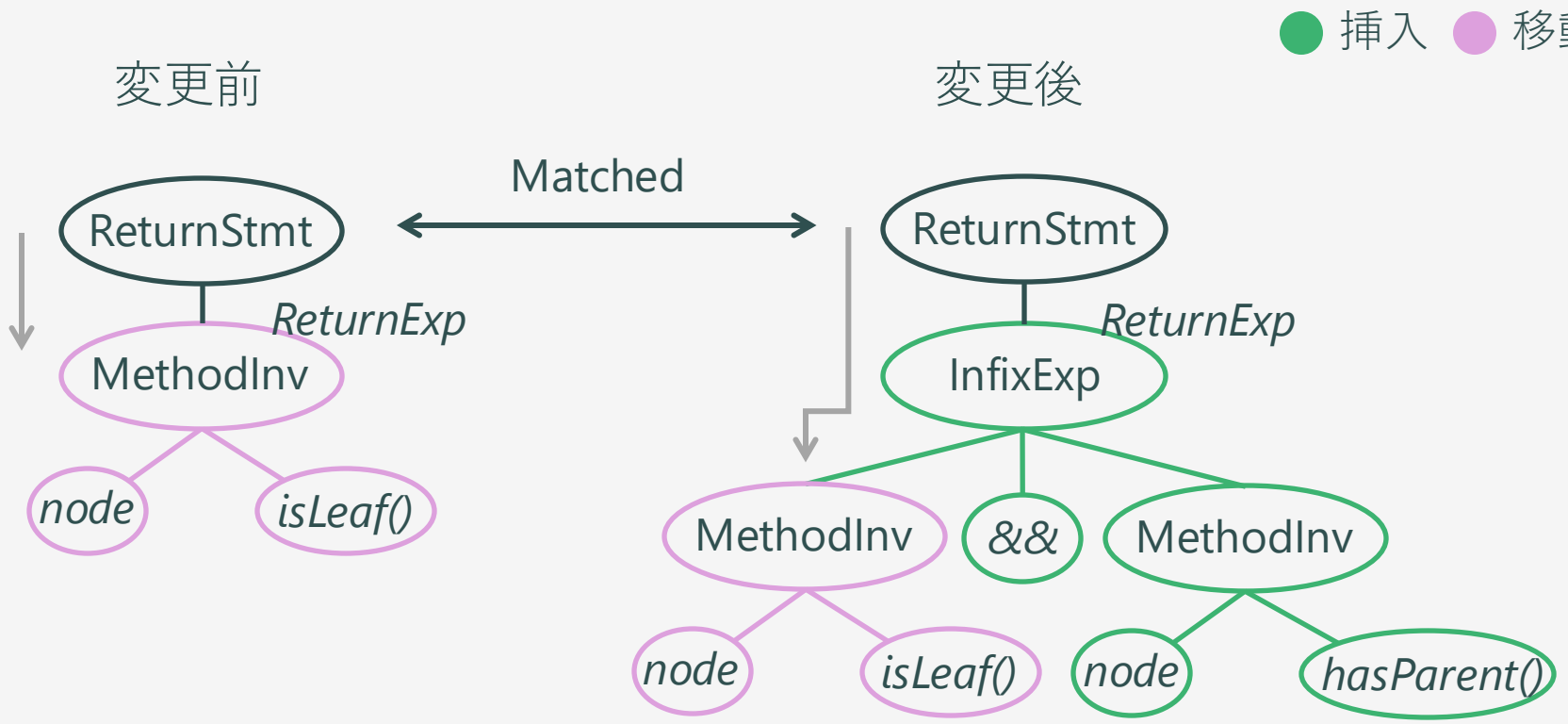
階層移動判定アルゴリズム



階層移動判定アルゴリズム

STEP1：調査した5種類に含まれる（条件2）
STEP2：同種類のノード内で完結する（条件1）
STEP3：横移動を含まない（条件3）

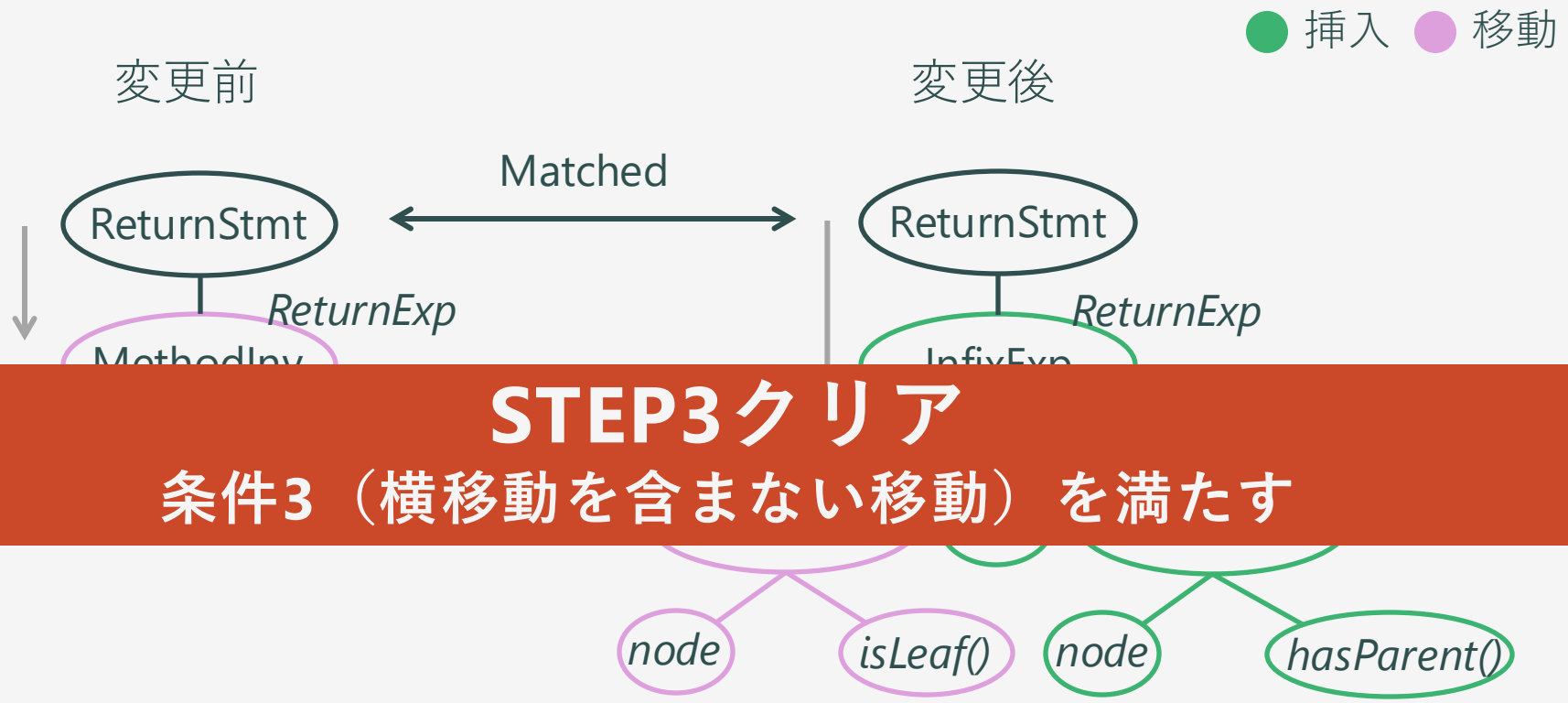
STEP3: 横移動を含まない（条件3）



経路上のノードを確認し異なる分岐をしていないかを判定

ルール1：挿入・削除は無視（ソースコード上ではトークン列の挿入・削除に見える）
ルール2：対応するノードの次の経路上のノードが同じ子要素かどうかを確認

STEP3: 横移動を含まない（条件3）



STEP3クリア

条件3（横移動を含まない移動）を満たす

ルール2

複数の子ノードが存在する場合に分岐が起こりうる

評価

比較対象

- 既存手法：GumTree

評価用データセット

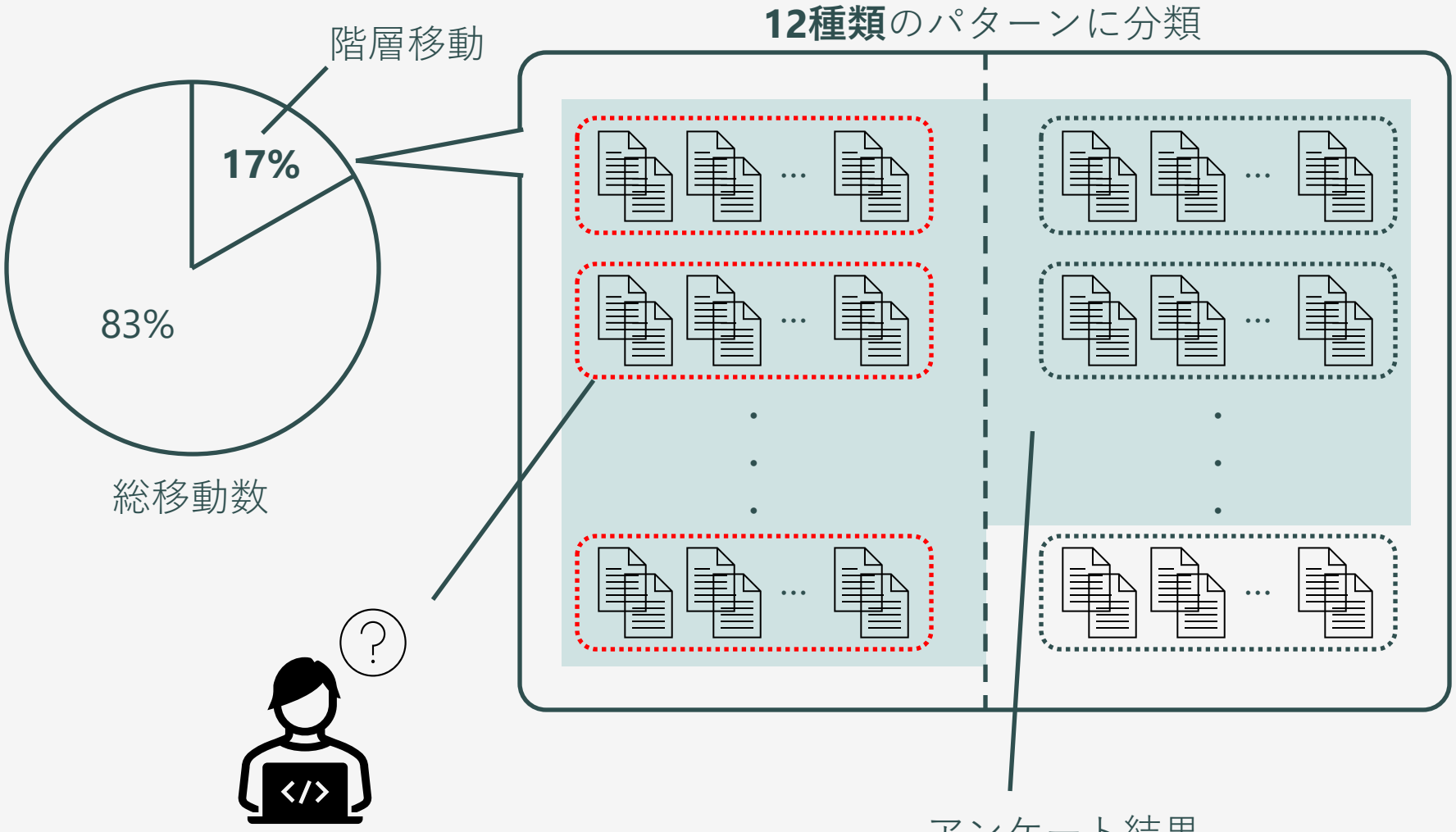
- 変更前後のJavaのソースコードの組：2045組
- バグ修正・機能追加

評価項目

- 検出した階層移動の数（自動評価）
- 階層移動の変更パターン（300組を対象に目視で分類）
- 既存手法が出力した階層移動の理解度（被験者評価*）
- 既存手法と提案手法の比較（被験者評価*）

* 被験者評価：CS専攻の学生10人を対象に実施

評価結果



被験者の**過半数**が既存手法が出力した
階層移動の原因を説明できない

アンケート結果
11種類の差分において
提案手法の方が理解しやすい

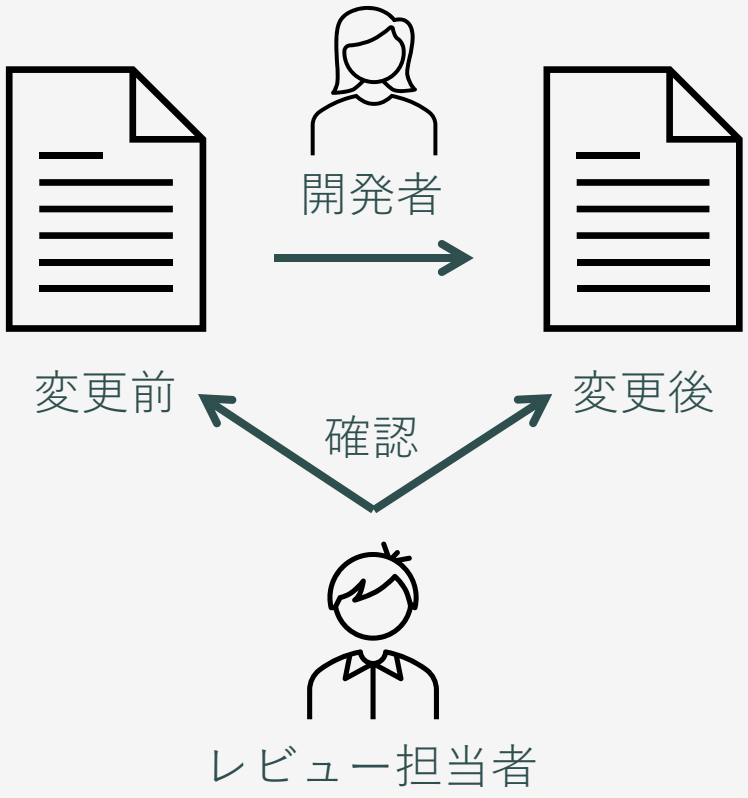
おわりに

- まとめ
  - ソースコード差分について
  - GumTreeの問題点（階層移動による認識の乖離）
  - 研究の目的・提案
  - 階層移動の定義（3条件）
  - 階層移動が起こりうるノードの調査
  - 階層移動の検出手法
  - 評価
- 今後の展望
  - Java以外のプログラミング言語への適用
  - JDT以外のAST生成ツールへの適用

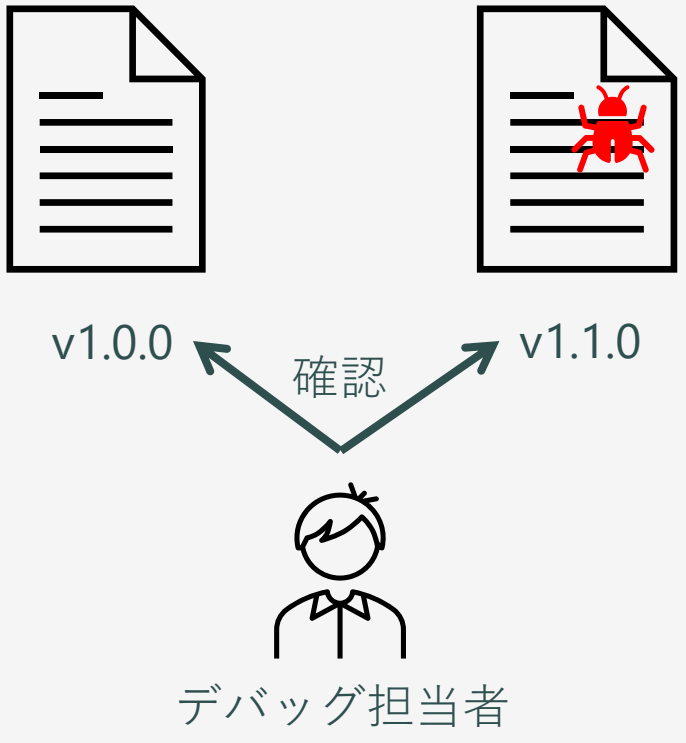
Appendix

ソースコード差分の用途

コードレビュー



デバッグ



既存の差分検出手法比較

テキストによる差分検出

テキストファイルで比較

コードの構造を考慮しない

e.g. Myersのアルゴリズム[1]

⭕実行時間が短い

❌差分の粒度が大きい（行）

❌検出できる操作（挿入 / 削除）

抽象構文木(AST)による差分検出

ASTで比較

コードの構造を考慮

e.g. GumTree[2][3]

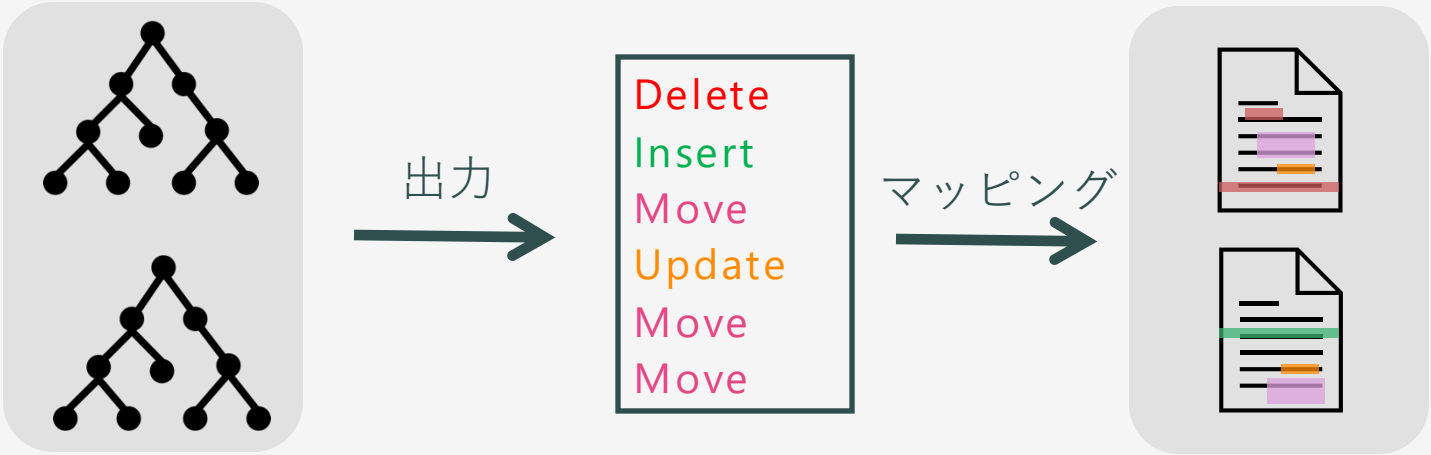
❌実行時間が長い

⭕差分の粒度が小さい（トークン）

⭕検出できる操作
（挿入 / 削除 / **更新** / **移動**）

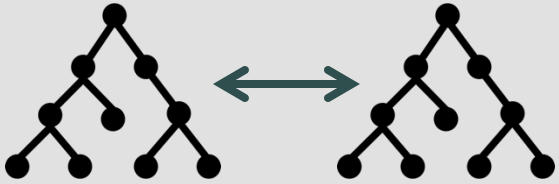
[1] E. W. Myers, “An o(nd) difference algorithm and its variations,” Algorithmica, vol. 1, pp. 251–266, 1986
[2] J. Falleri etc. “Finegrained and accurate source code differencing” in Proc. of the 29th ACM/IEEE ASE, 2014, pp. 313–324
[3] J. Falleri etc. “Fine-grained, accurate and scalable source differencing” in Proc. of the 46th IEEE/ACM ICSE. 2024, pp. 1-12

GumTreeの差分検出方法



抽象構文木間の差分　　編集スクリプト　　ソースコード上の差分

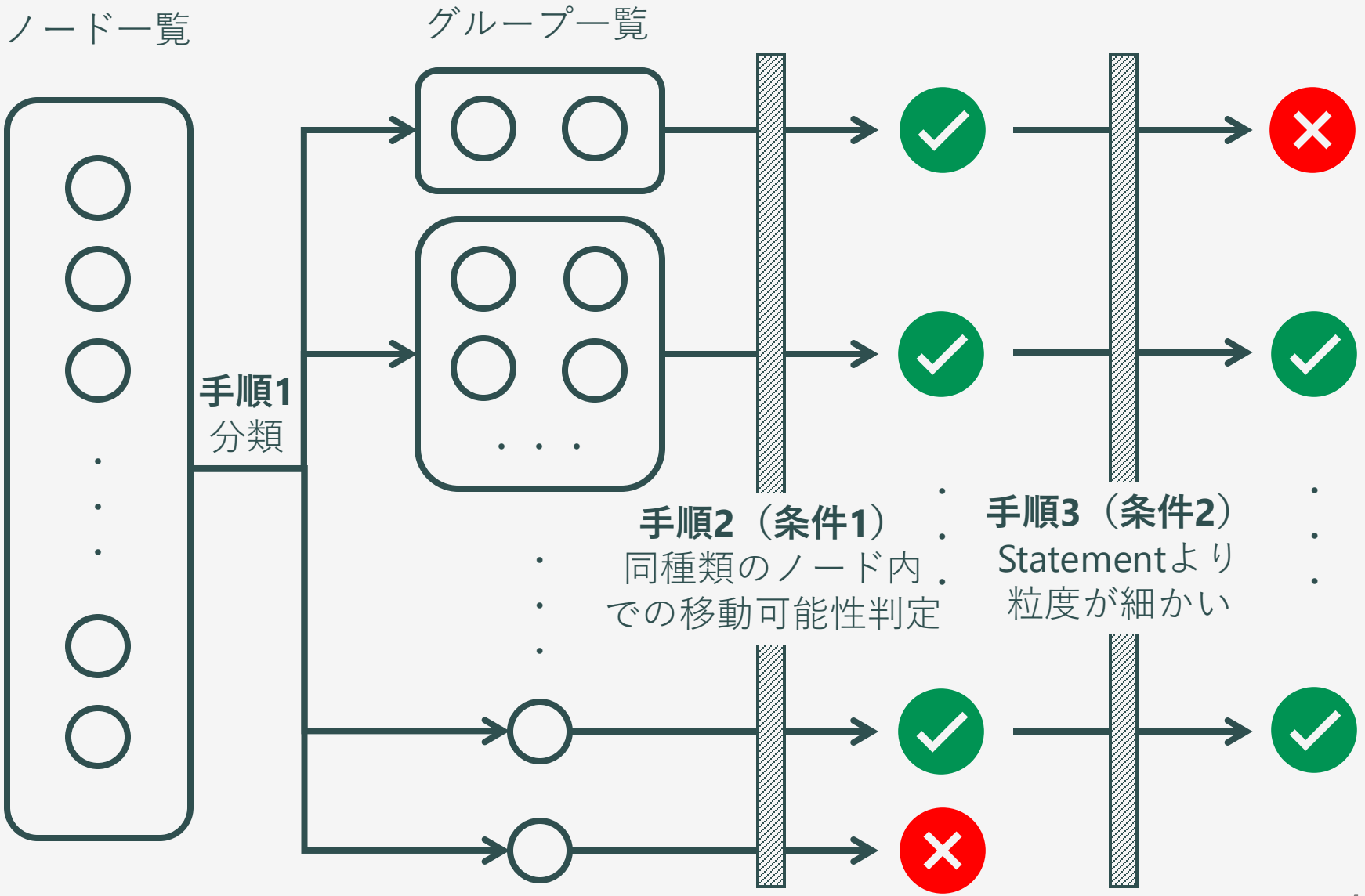
差分計算アルゴリズム



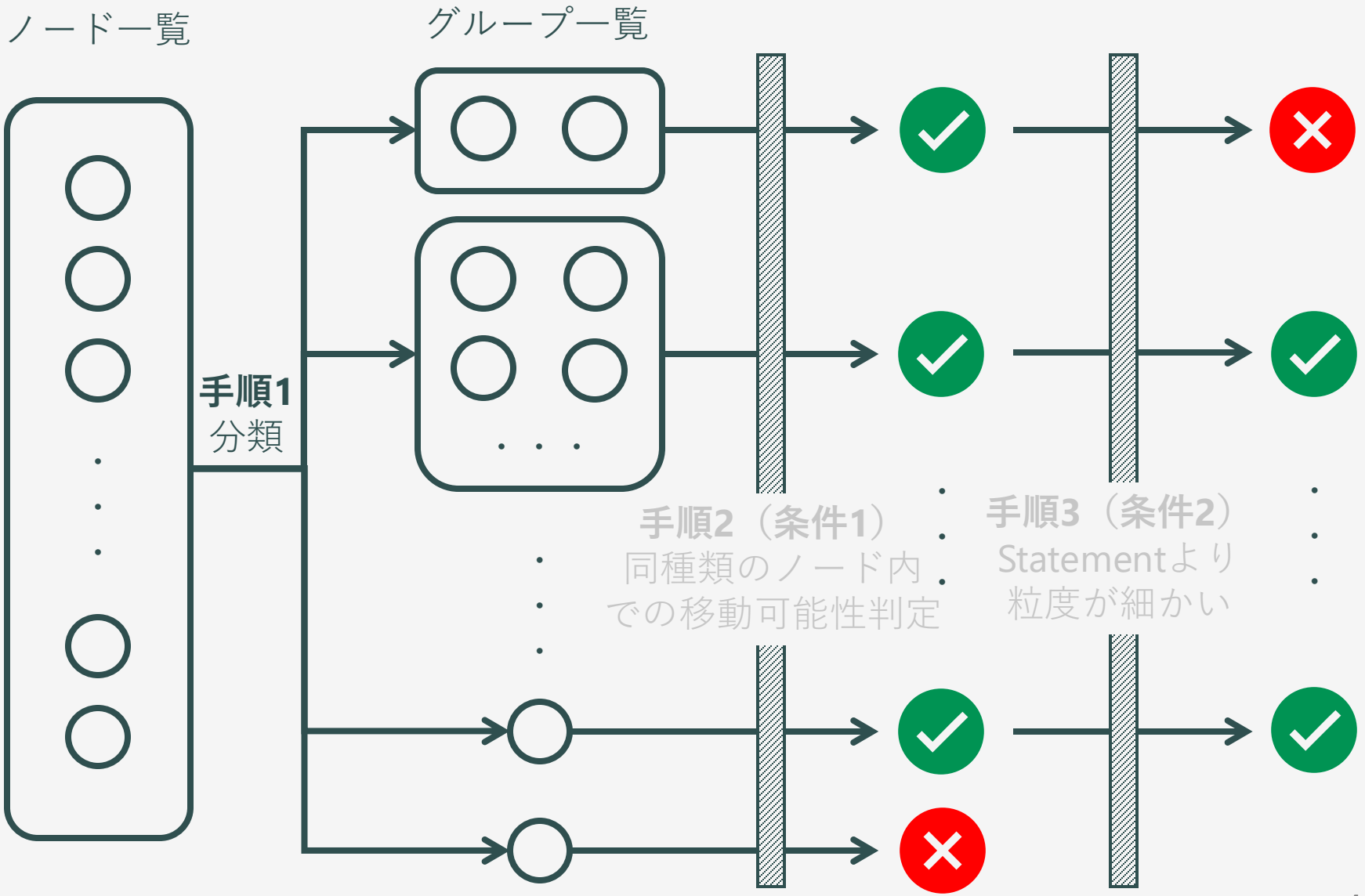
変更前　　変更後

1. 変更前後で対応するノードをマッチ
2. 対応するノードがない→**削除** or **挿入**
3. 対応するノード同士で値が変化→**更新**
4. 対応するノード同士で親ノードが変化→**移動**

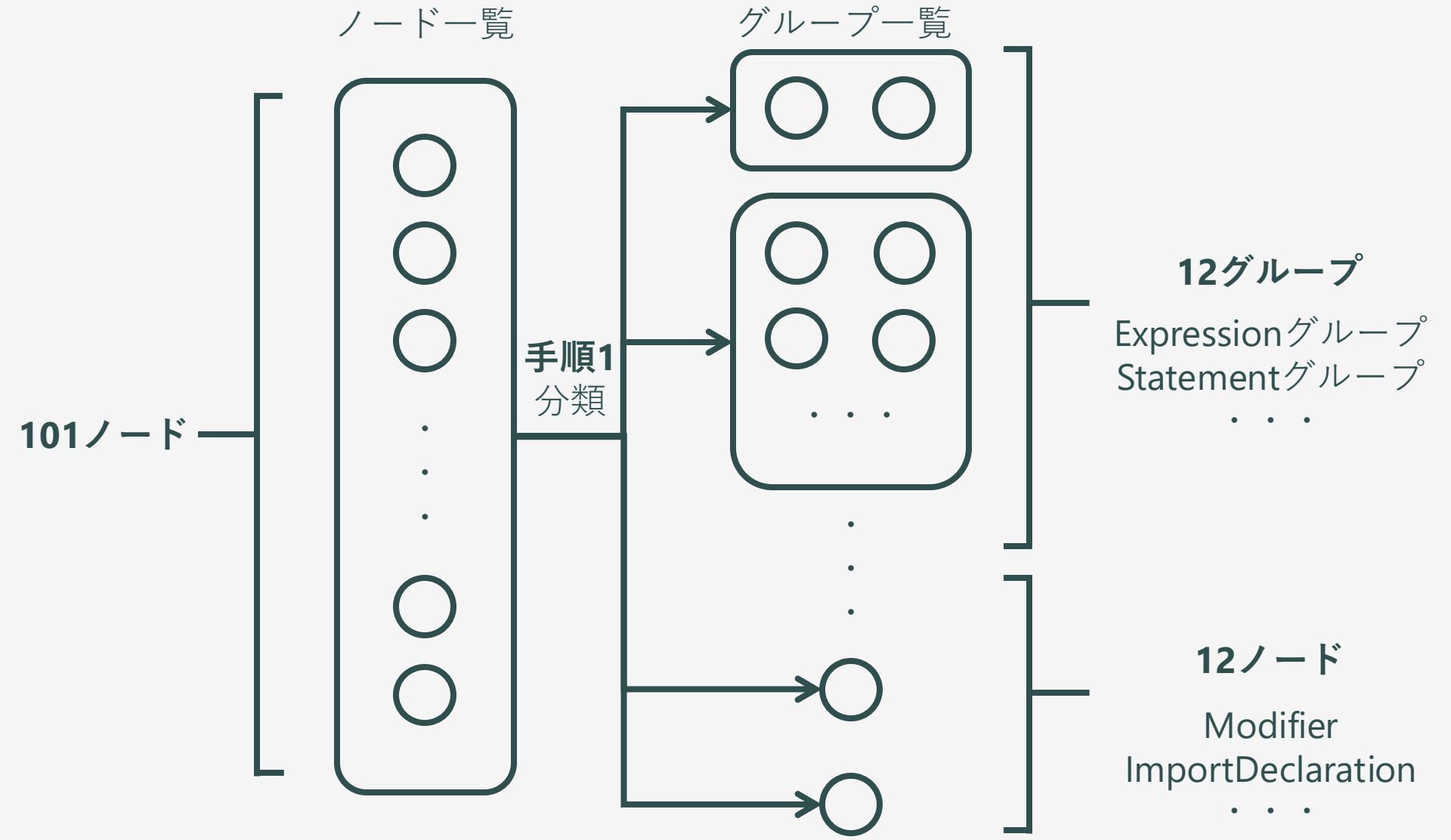
階層移動が起こりうるノードの調査手順



階層移動が起こりうるノードの調査手順

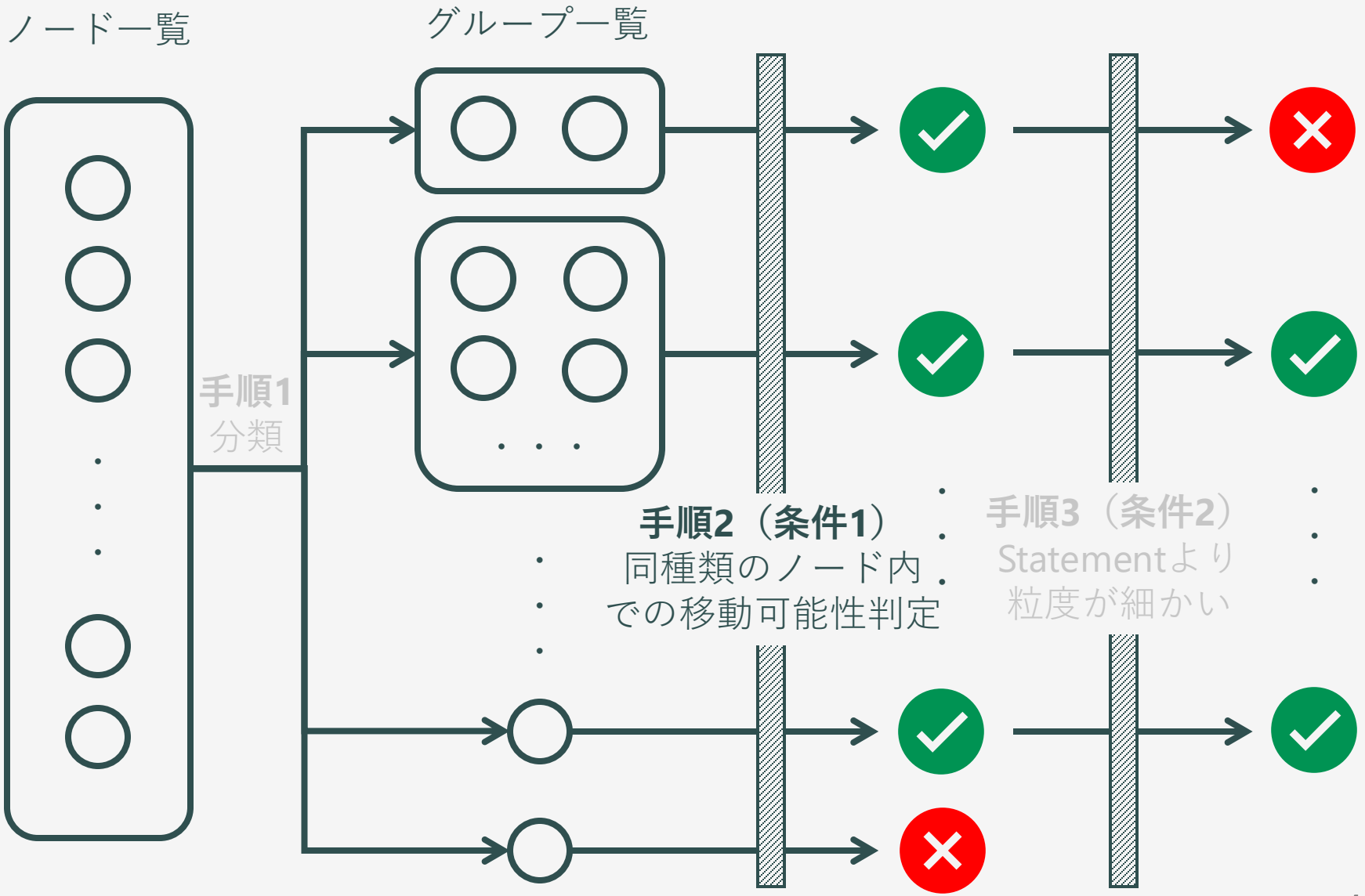


手順1: ノードの分類



JDTの実装におけるASTノードクラスの継承関係を参考

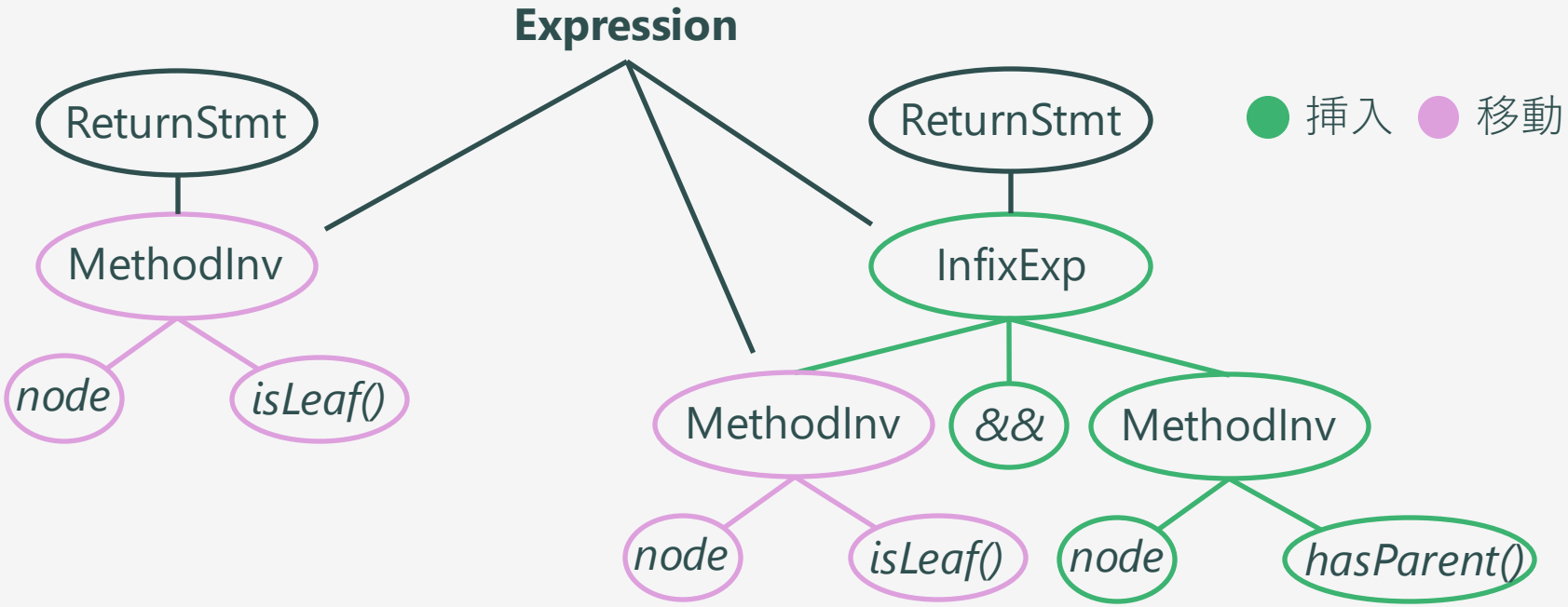
階層移動が起こりうるノードの調査手順



手順2: 同種類のノード内での移動発生可能性の判定方法

自分が所属するグループのノードを子ノードに持つかどうか

$$InfExp\ ::=\ Exp\ Operator\ Exp\ (Operator\ Exp)^*$$

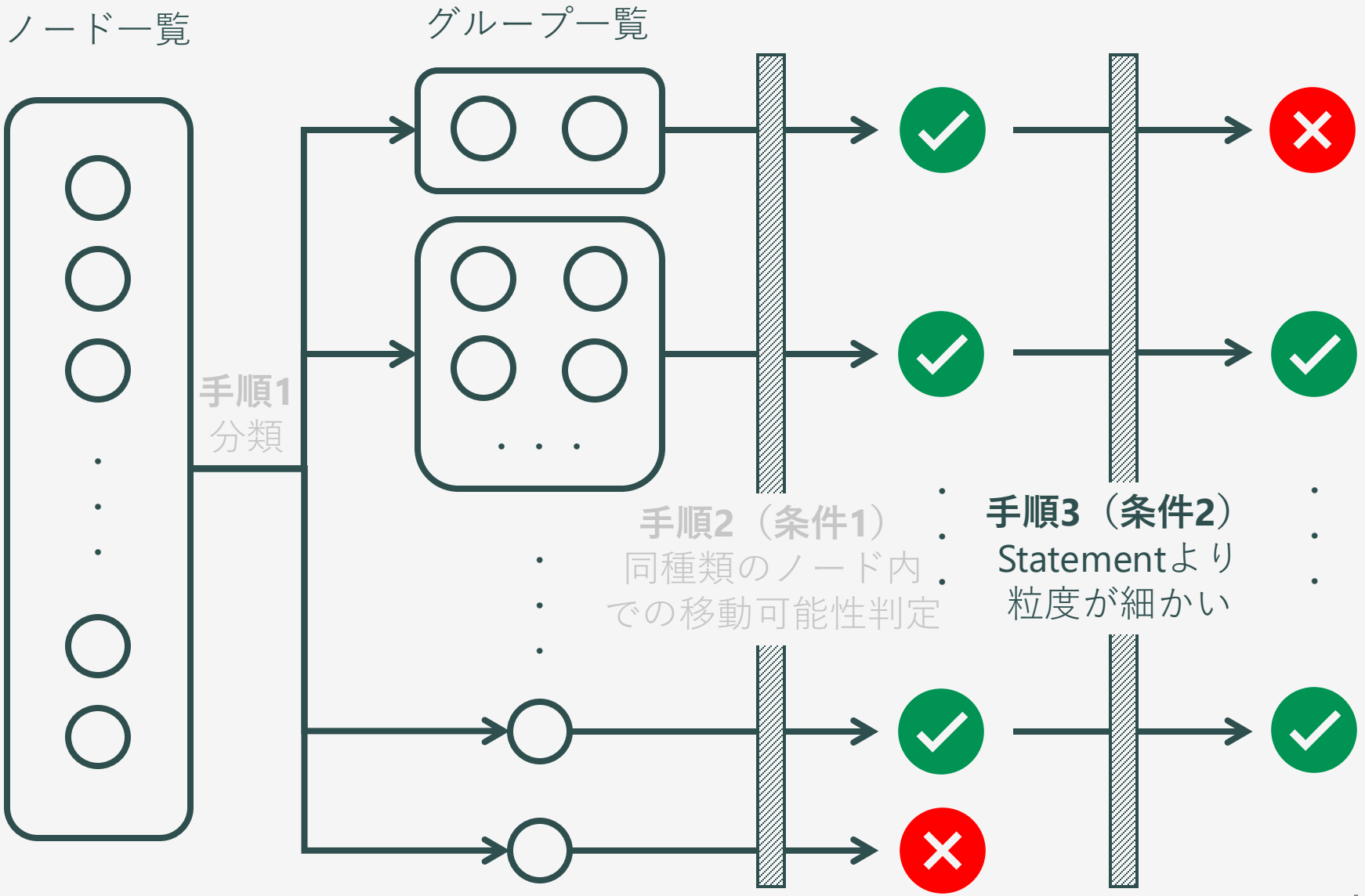


グループに1つでもそのようなノードがあれば検出対象とする

調査結果

BodyDeclaration Statement Expression Name Pattern Type

階層移動が起こりうるノードの調査手順



手順3: Statementよりも粒度が細かいノード

手順2の調査結果

~~BodyDeclaration~~ ~~Statement~~ **Expression** **Name** **Pattern** **Type**

例外として**if文のelse構造**における階層移動も検出対象とする

手順3の調査結果

Expression **Name** **Pattern** **Type** **if文のelse構造**

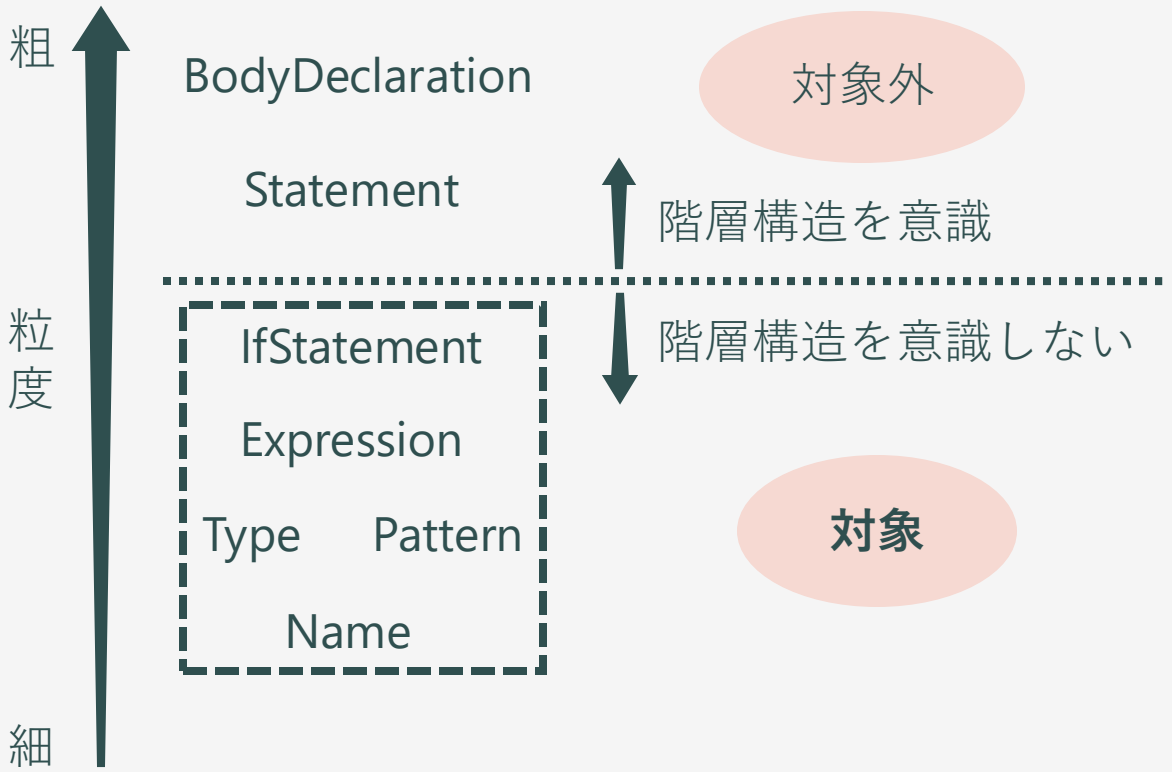
5種類に含まれるノード以外の移動は確認しなくて良い

条件1: 同種類のノード内階層移動可能性判定

$$InfExp ::= Exp\ Operator\ Exp\ (Operator\ Exp)^*$$

グループ名	可能性のあるノード数	全ノード数
BodyDeclaration	2	10
Statement	10	23
Expression	24	41
Name	1	3
Pattern	2	5
Type	6	9

（3）Statementより粒度の細かいノード

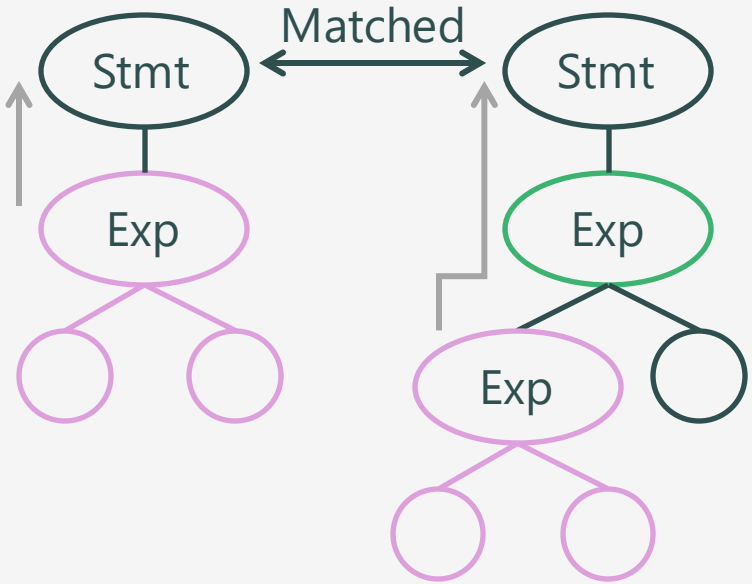


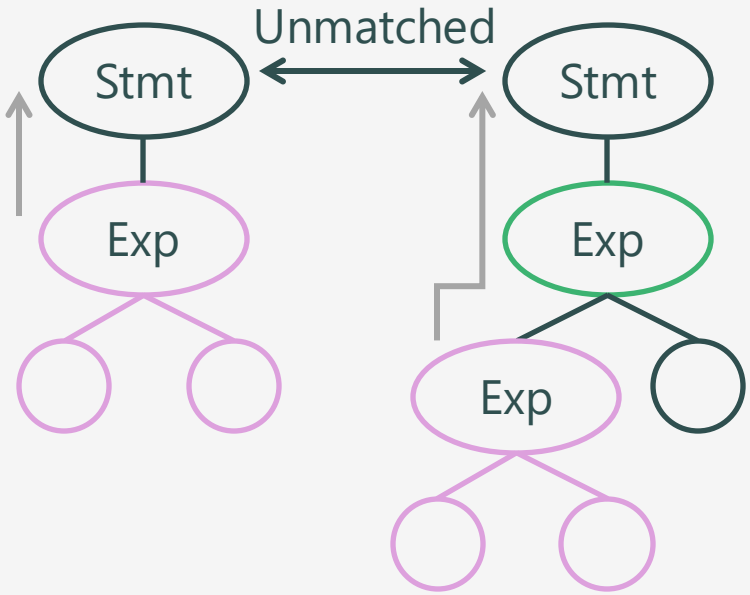
※ 例外としてIfStatementのelse構造は対象とする

条件1の判定（同じ種類のノード間での移動）

先祖ノードの一致判定

同種類のノードが続く限り遡る

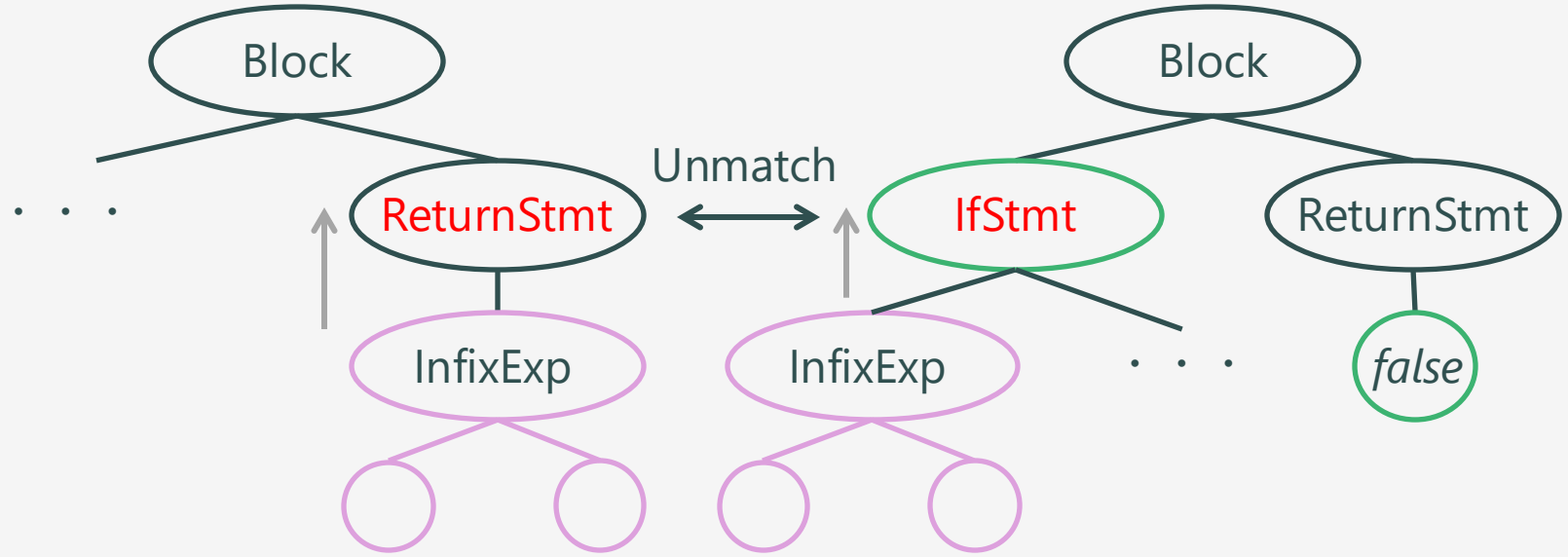




階層移動の可能性**あり**

異なるノードに横移動した
階層移動の可能性**なし**

条件1の判定例



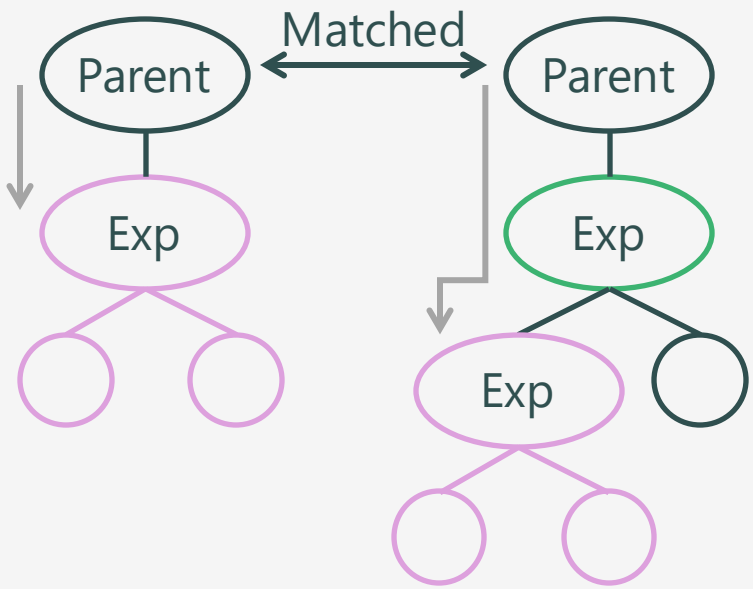
異種類のノードに辿り着いた時に

マッチしていない場合は異種類のノードを跨いだ移動

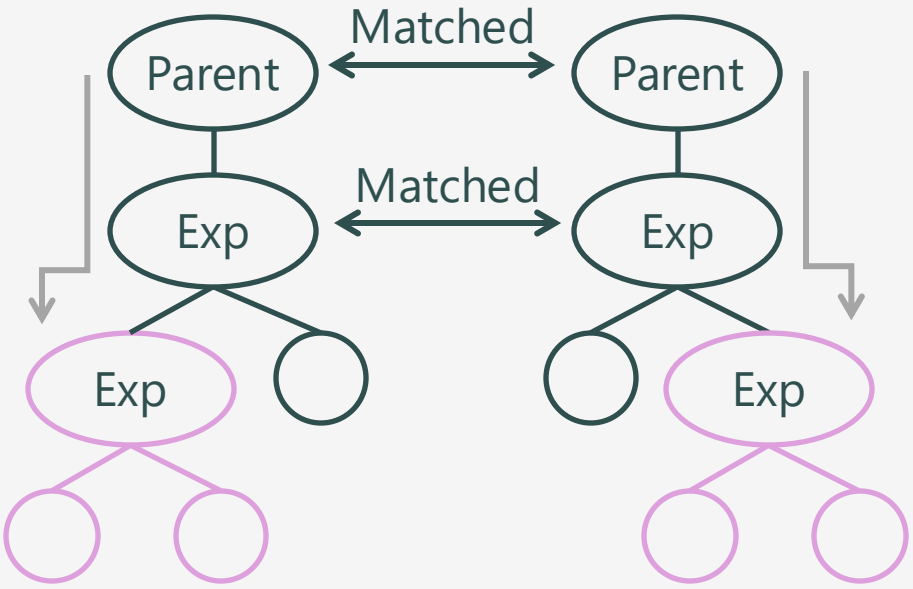
条件3の判定（横移動を含まない上下移動）

経路比較

経路を比較し横移動が発生していないか確認する



横移動なし
階層移動の可能性**あり**

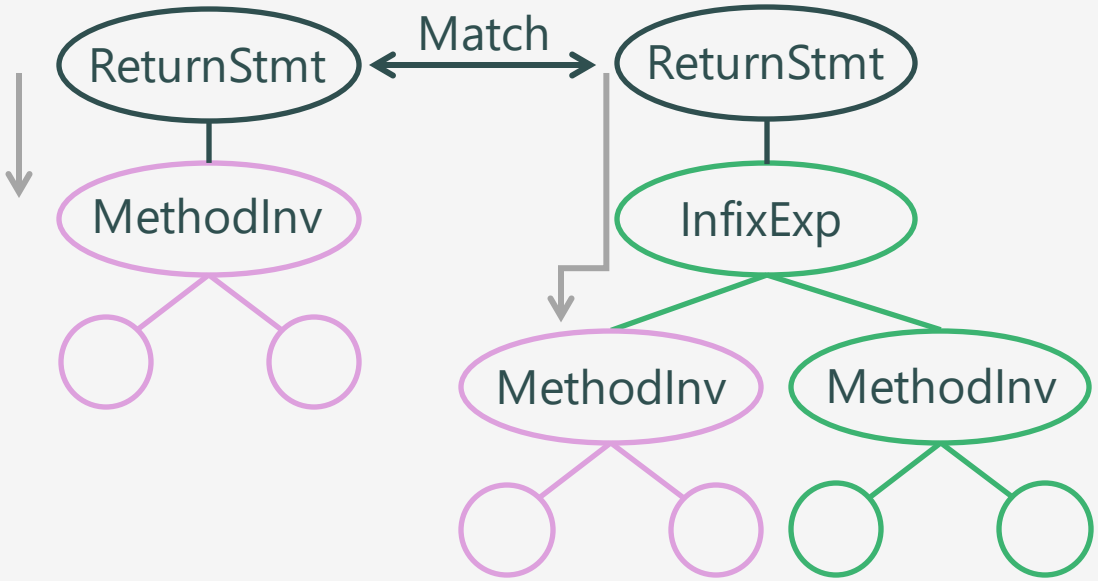


横移動あり
階層移動の可能性**なし**

経路判定ルール1

ルール1：両方の経路上に存在しないノードは無視

経路上におけるノードの追加/削除は横移動に関係なし



ソースコード上では前後にコード片が追加されただけ

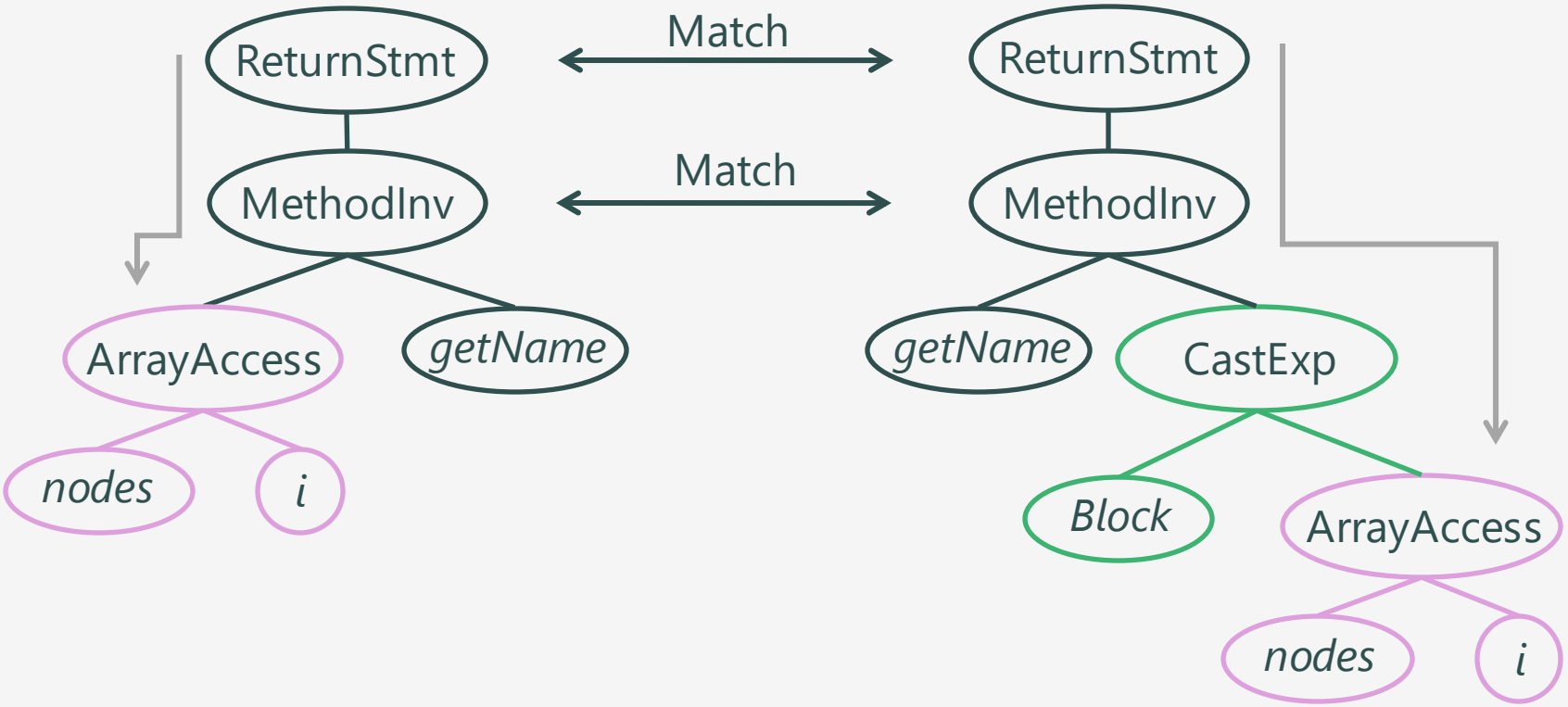
```
return
  node.isLeaf();
```

```
return
  node.isLeaf() && node.hasParent();
```

経路判定ルール2-1

ルール2-1：両経路にノードのマッチがある場合

⚠ノードだけ見ても横移動は判定できない



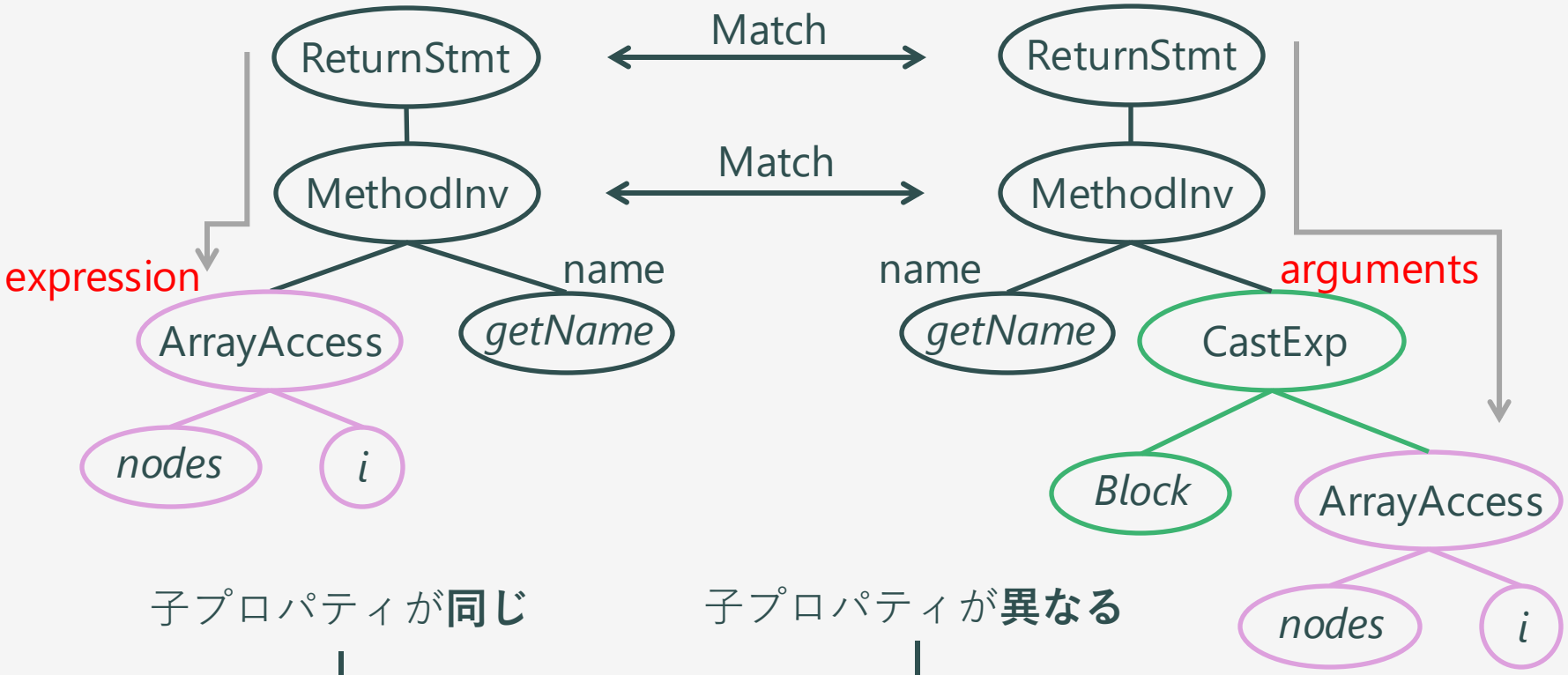

```
return
  nodes[i].getName();
```

```
return
  getName((Block) nodes[i]);
```

経路判定ルール2-1

ルール2-1：両経路にノードのマッチがある場合

経路上の次のノードが同じ子プロパティを持つか判定する



子プロパティが**同じ**

↓

階層移動の可能性**あり**

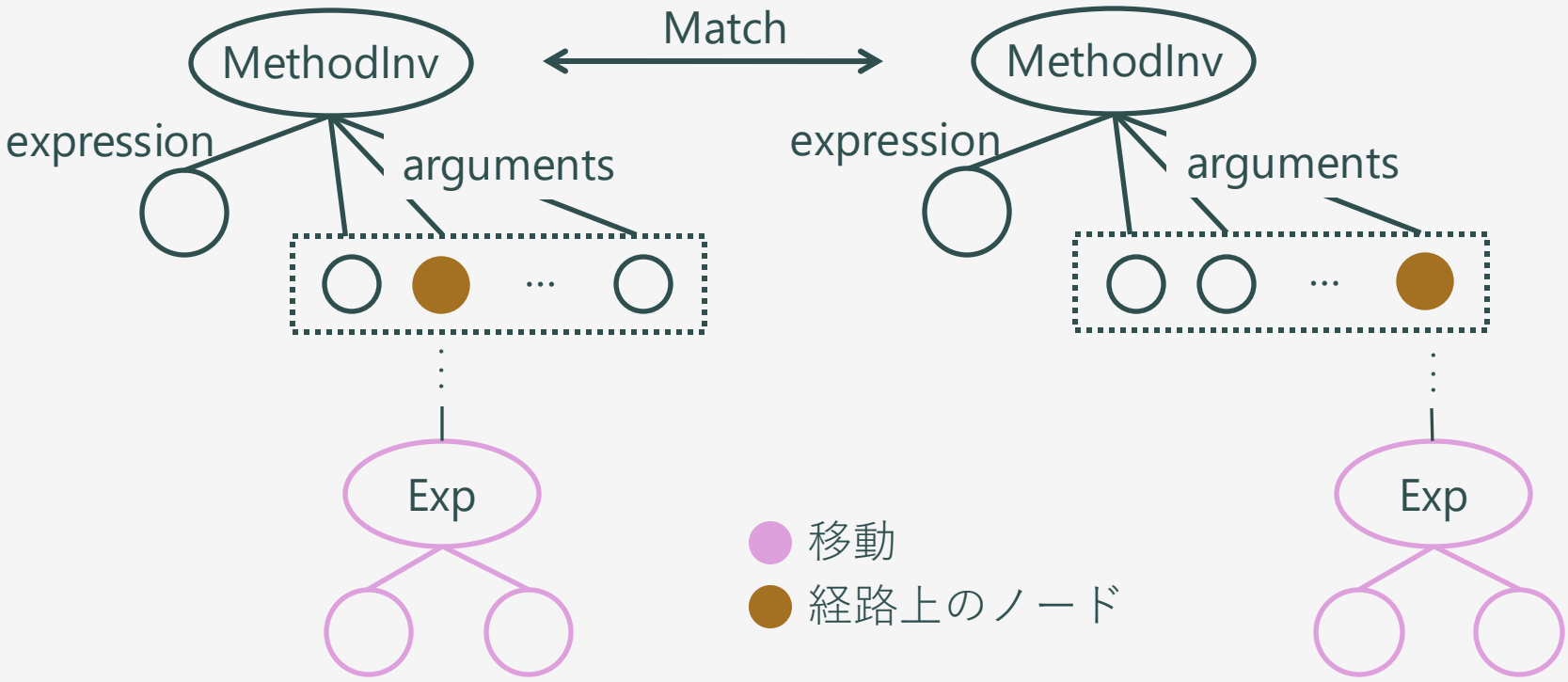
子プロパティが**異なる**

↓

横移動あり
階層移動の可能性**なし**

経路判定ルール2-2

ルール2-2：同じ子プロパティが複数存在する場合



同じ子プロパティ内での**横移動なし**

↓

階層移動の可能性**あり**

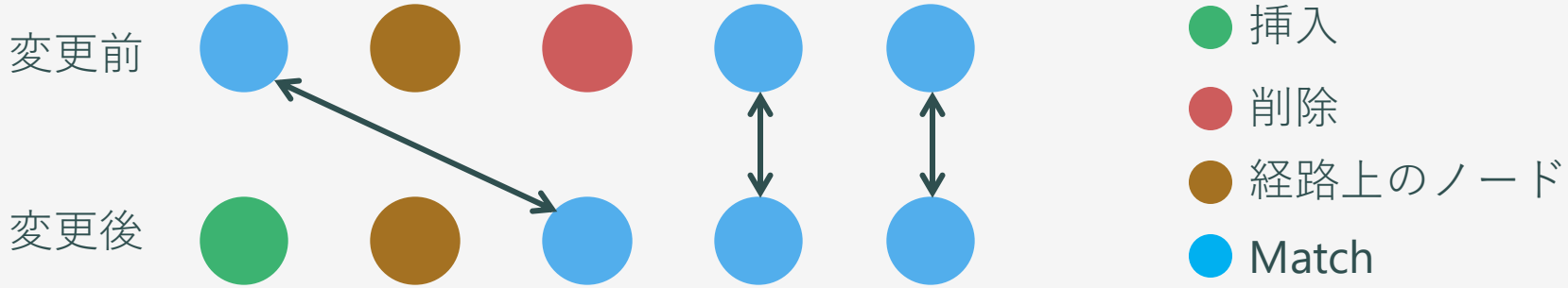
同じ子プロパティ内での**横移動あり**

↓

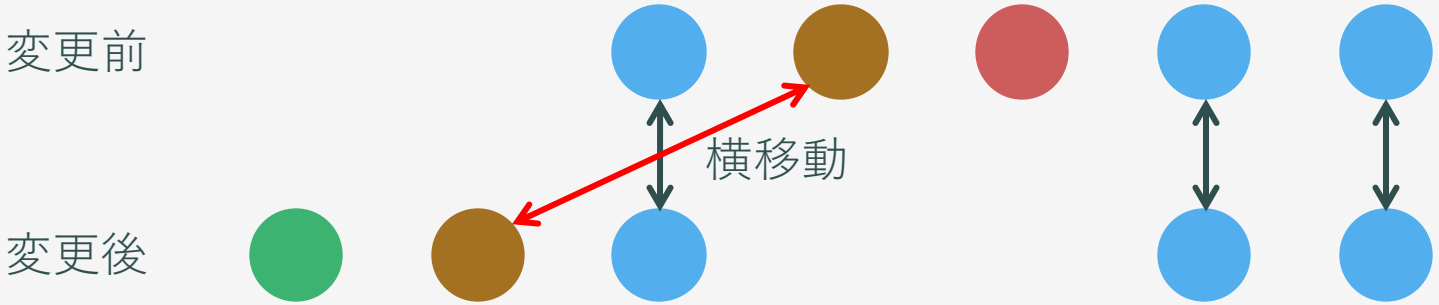
階層移動の可能性**なし**

同じ子プロパティ内での横移動検出

同プロパティ内ノードのマッチでLCSを取得



LCSでマッチしたノードを基準にずれがあるかを判定



評価結果

検出した階層移動の数

- 総移動数の内**17%**が階層移動（ 3172個中540個）

階層移動の変更パターン

- 手作業で**12種類**に分類

既存手法が出力した差分の理解度

- 従来のGumTreeが出力した階層移動の内**6種類**について過半数の学生が移動の原因を理解できなかった

提案手法が出力した差分の理解しやすさ

- **11種類**について提案手法が出力した差分の方が理解しやすいというアンケート結果

階層移動による差分情報は開発者にとって不要である

階層移動の個数・割合

階層移動の検出数

- 対象：計2045個のコミット（データセット全体）
- 階層移動を検出した組数：2045組中348組（17.0%）
- 階層移動数：3172個中540個（17.0%）
- ノードの種類毎の階層移動数
  - Expression: 446個（82.6%）
  - IfStatement: 59個（10.9%）
  - Name: 23個（4.3%）
  - Type: 12個（2.2%）
  - Pattern: 0個（0%）

階層移動の変更パターン毎の分類結果

パターン名	概要	個数
Expression-1	四則演算や論理演算での移動	41
Expression-2	メソッドの引数から（へ）の移動	34
Expression-3	メソッドチェーンの追加・削除による移動	15
Expression-4	丸括弧の追加・削除による移動	11
Expression-5	否定演算子の追加・削除による移動	6
Expression-6	キャスト演算子の追加・削除による移動	6
Expression-7	三項演算の要素から（へ）の移動	6
Name-1	ドットで連結した名前の追加・削除による移動	4
Type-1	配列を表すブラケットの追加による移動	1
Type-2	他の型の要素型への移動	1
If-1	既存のif文が，新しく追加された if 文の後のelse節へ移動	7
If-2	既存のif文に対するelse節の追加・削除による移動	2

四則演算や論理演算での移動

既存手法による差分出力

変更前

```
raw == String.class || raw == Object.class
```

変更後

```
raw == String.class || raw == Object.class || raw == CharSequence.class
```

提案手法による差分出力

変更前

```
raw == String.class || raw == Object.class
```

変更後

```
raw == String.class || raw == Object.class || raw == CharSequence.class
```

メソッドの引数から（へ）の移動

既存手法による差分出力

変更前

```
"" + eachArg.charAt(i)
```

変更後

```
String.valueOf(eachArg.charAt(i))
```

提案手法による差分出力

変更前

```
"" + eachArg.charAt(i)
```

変更後

```
String.valueOf(eachArg.charAt(i))
```

メソッドチェーンの追加・削除による移動

既存手法による差分出力

変更前

```
DEFAULT.withIgnoreEmptyLines(false)
```

変更後

```
DEFAULT.withIgnoreEmptyLines(false).withAllowMissingColumnNames(true)
```

提案手法による差分出力

変更前

```
DEFAULT.withIgnoreEmptyLines(false)
```

変更後

```
DEFAULT.withIgnoreEmptyLines(false).withAllowMissingColumnNames(true)
```

丸括弧の追加・削除による移動

既存手法による差分出力

変更前

```
(double) (getSampleSize() * getNumberOfSuccesses())
                                / (double) getPopulationSize()
```

変更後

```
getSampleSize() *
            (getNumberOfSuccesses() / (double) getPopulationSize())
```

提案手法による差分出力

変更前

```
(double) (getSampleSize() * getNumberOfSuccesses())
                                / (double) getPopulationSize()
```

変更後

```
getSampleSize() *
            (getNumberOfSuccesses() / (double) getPopulationSize())
```

否定演算子の追加・削除による移動

既存手法による差分出力

変更前

```
tag.isSelfClosing()
```

変更後

```
!tag.isEmpty()
```

提案手法による差分出力

変更前

```
tag.isSelfClosing()
```

変更後

```
!tag.isEmpty()
```

キャスト演算子の追加・削除による移動

既存手法による差分出力

変更前

```
(Option) options.getOption(arg)
```

変更後

```
options.getOption(ch)
```

提案手法による差分出力

変更前

```
(Option) options.getOption(arg)
```

変更後

```
options.getOption(ch)
```

三項演算の要素から（へ）の移動

既存手法による差分出力

変更前

```
return
  TypeHandler.createValue(res, value);
```

変更後

```
return
  res == null ? null : TypeHandler.createValue(res, value);
```

提案手法による差分出力

変更前

```
return
  TypeHandler.createValue(res, value);
```

変更後

```
return
  res == null ? null : TypeHandler.createValue(res, value);
```

ドットで連結した名前の追加・削除による移動

既存手法による差分出力

変更前

```
import
  com.ning.billing.entitlement.api.user.ISubscription;
```

変更後

```
import static
  com.ning.billing.entitlement.api.user.Isubscription.SubscriptionState;
```

提案手法による差分出力

変更前

```
import
  com.ning.billing.entitlement.api.user.ISubscription;
```

変更後

```
import static
  com.ning.billing.entitlement.api.user.Isubscription.SubscriptionState;
```

配列を表すブラケットの追加による移動

既存手法による差分出力

変更前

```
private final Constraint constraint;
```

変更後

```
private final Constraint[] constraints;
```

提案手法による差分出力

変更前

```
private final Constraint constraint;
```

変更後

```
private final Constraint[] constraints;
```

他の型の要素方への移動

既存手法による差分出力

変更前

```
private final HashMap<String, String>
                                namespaces = new HashMap<>();
```

変更後

```
private final Stack<HashMap<String, String>>
                                namespacesStack = new Stack<>();
```

提案手法による差分出力

変更前

```
private final HashMap<String, String>
                                namespaces = new HashMap<>();
```

変更後

```
private final Stack<HashMap<String, String>>
                                namespacesStack = new Stack<>();
```

変更前

```
if (options.hasOption(arg.substring(0, 2)))
{
    ...
}
else
{
    ...
}
```

変更後

```
if (opt.indexOf('=') != -1 && ...)
{
    ...
}
else if (options.hasOption(arg.substring(0, 2)))
{
   ...
}
else
{
    ...
}
```

変更前

```
if (options.hasOption(arg.substring(0, 2)))
{
    ...
}
else
{
    ...
}
```

変更後

```
if (opt.indexOf('=') != -1 && ...)
{
    ...
}
else if (options.hasOption(arg.substring(0, 2)))
{
   ...
}
else
{
    ...
}
```

変更前

```
if (token == JsonToken.FIELD_NAME) {
    ...
} else if (token == JsonToken.START_OBJECT) {
    ...
} else if (token == JsonToken.VALUE_TRUE ...) {
    ...
} else {
    ...
}
```

変更後

```
if (token == JsonToken.FIELD_NAME) {
    ...
} else if (token == JsonToken.START_OBJECT) {
    ...
} if (token == JsonToken.START_ARRAY) {
    ...
} else if (token == JsonToken.VALUE_TRUE ...) {
    ...
} else {
    ...
}
```

変更前

```
if (token == JsonToken.FIELD_NAME) {
    ...
} else if (token == JsonToken.START_OBJECT) {
    ...
} else if (token == JsonToken.VALUE_TRUE ...) {
    ...
} else {
    ...
}
```

変更後

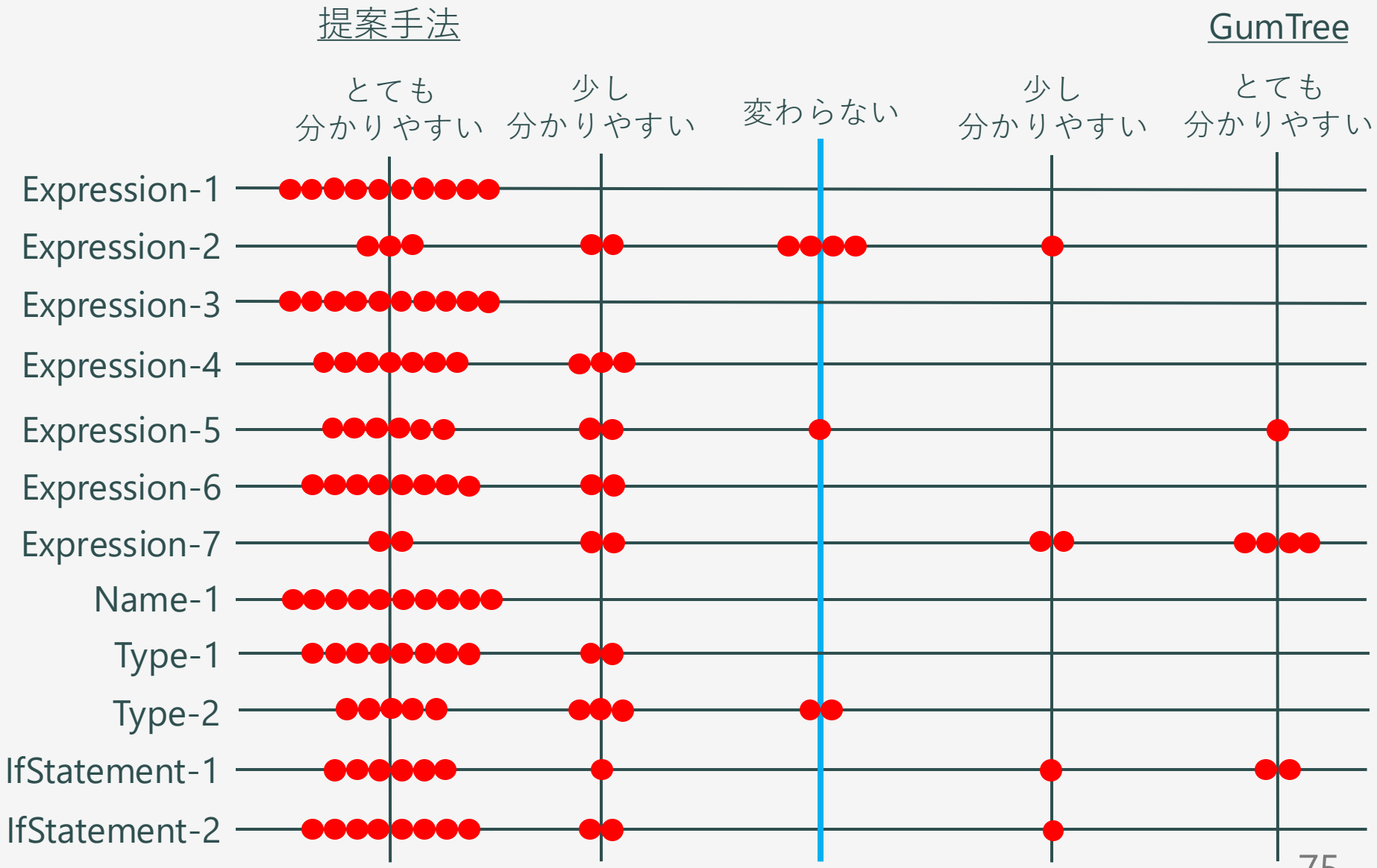
```
if (token == JsonToken.FIELD_NAME) {
    ...
} else if (token == JsonToken.START_OBJECT) {
    ...
} if (token == JsonToken.START_ARRAY) {
    ...
} else if (token == JsonToken.VALUE_TRUE ...) {
    ...
} else {
    ...
}
```

階層移動の原因を理解できなかった人数

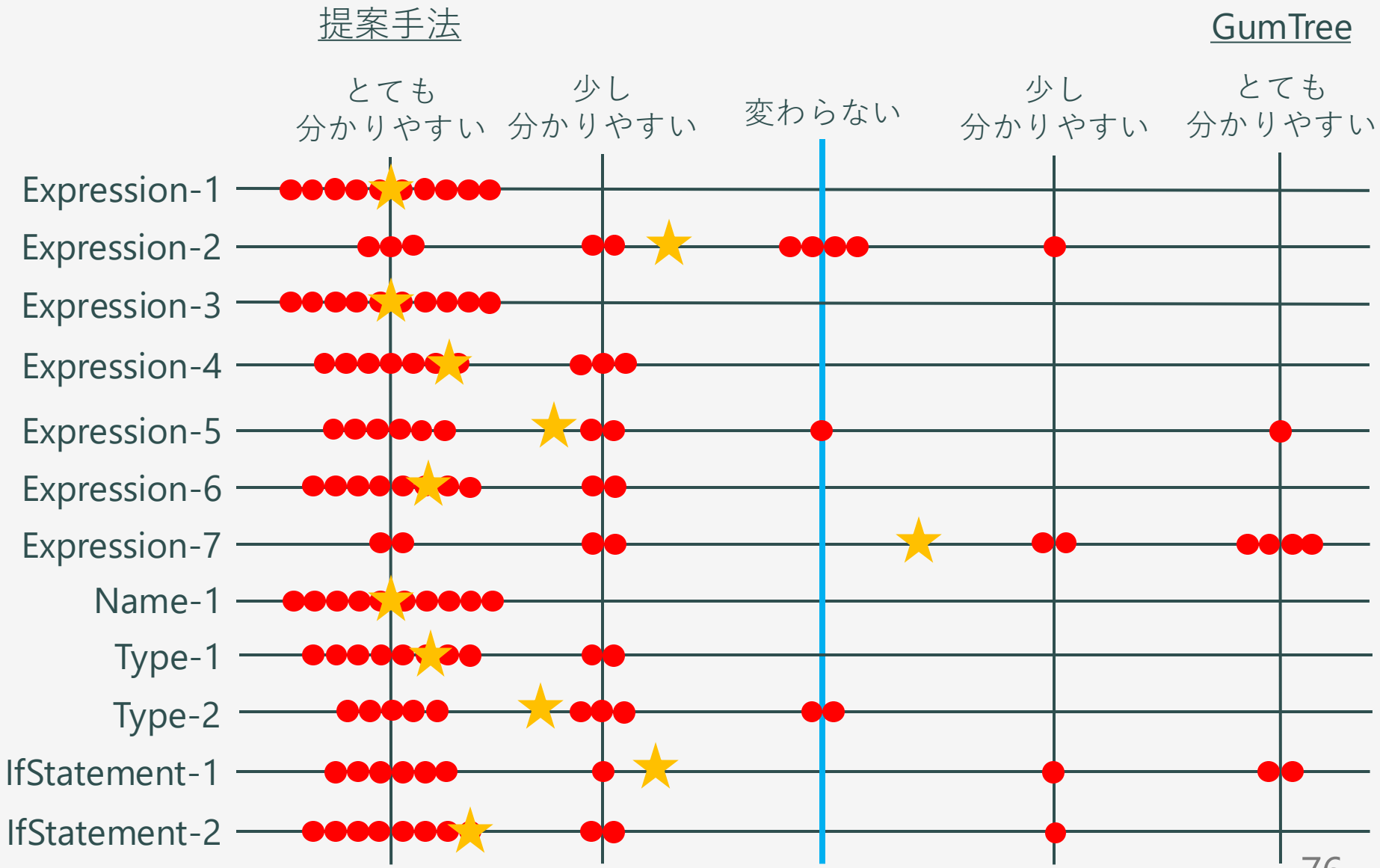
パターン名	理解している	理解していない
Expression-1	3	7
Expression-2	9	1
Expression-3	3	7
Expression-4	6	4
Expression-5	6	4
Expression-6	4	6
Expression-7	9	1
Name-1	1	9
Type-1	3	7
Type-2	9	1
If-1	6	4
If-2	4	6

半数の種類の階層移動において
過半数が移動の原因を理解できなかった

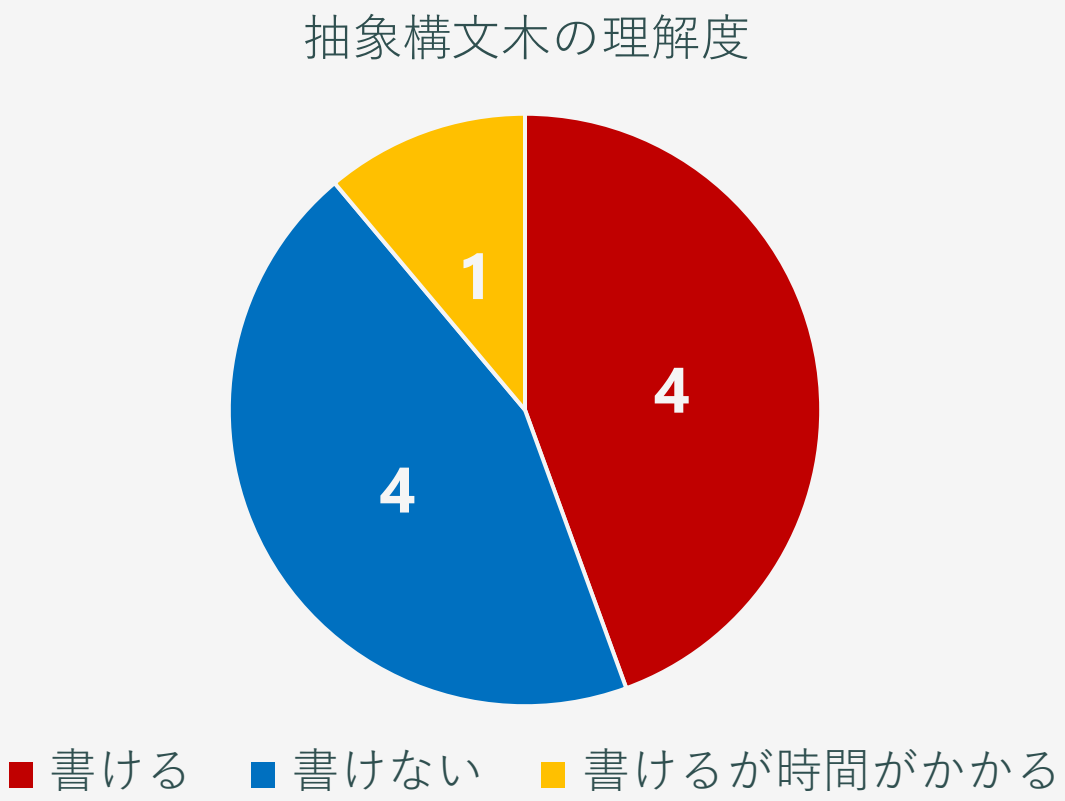
既存手法 vs 提案手法 アンケート結果（1/2）



既存手法 vs 提案手法 アンケート結果（2/2）



ASTの理解度調査結果



被験者の過半数が簡単な抽象構文木をすぐに書けなかった

妥当性への脅威

- 提案手法の適用対象が限定的である
  - Java，Eclipse JDT Coreに限定
- 3条件が全ての不要な階層移動を網羅できていない
  - 条件の決め方で検出できる階層移動は変わりうる
  - 出来るだけ多くの差分を確認した（著者の独断）
  - 確度の高い階層移動のみを対象にした
- 階層移動の分類が属人的である（著者の独断）
- 被験者実験の対象が限定的である
  - 各パターンを代表する差分しか評価できていない
  - 被験者が学生である

おわり