

AIDING CODE CHANGE UNDERSTANDING WITH SEMANTIC CHANGE IMPACT ANALYSIS

QUINN HANAM, ALI MESBAH, REID HOLMES

従来のコードレビュー支援ツール

▶ 変更影響分析

- UNIX DIFF：変更（追加・削除）された行を表示
 - 変更の影響範囲はわからない

問題点

制御・データ依存関係情報から
コード変更の影響をトレースできない

コード変更の影響分析

▶ 静的変更影響分析

(SYNCIA : Syntax Change Impact Analysis)

- ソースコードの変更に伴う動作への影響
- UNIX DIFFを使用して変数とデータを識別
- 制御依存関係は範囲内のすべての変数に対して計算

コード変更の理解タスクを実行する開発者の
パフォーマンスを向上できるかどうか調査

予備調査

静的変更影響分析手法はUNIX DIFFと比べて
改善できることはあるのか？

- ▶ UNIX DIFF
 - ▶ SYNCIA : UNIX DIFFと制御, データ依存情報
- ✓ 2通りでコードレビューを実施

予備調査

- ▶ 学部生 8 名と企業開発者 3 名
- ▶ 134個のNode.jsプロジェクトのランダムな 1 0 個のコミットを抽出
 - 正確に1つのファイルが変更
 - UNIX DIFFによって10～30行が挿入されたと判断
- ▶ 使用するコミットをランダムに3つ選択

レビュータスク

- ▶ 4 種類タスクを含むレビュータスクを実施
 - スコープ, 分岐条件, 引数, 関数呼び出し位置

結果

Task	Subject	Mean Search Time (s)		Success Rate (%)	
		UNIXDIFF	SYNCIA	UNIXDIFF	SYNCIA
Scope	Karma	420	-122	0	50
	PM2	360	+60	60	0
	Popcorn	136	+77	100	100
Condition	Karma	147	-39	75	100
	PM2	136	+77	100	100
	Popcorn	90	+111	100	100
Arguments	Karma	290	-24	67	100
	PM2	284	-33	50	67
	Popcorn	146	-48	100	100
Callsites	Karma	272	-54	100	100
	PM2	349	+14	29	42
	Popcorn	296	-141	100	100

レビュータスク

- ▶ 4 種類タスクを含むレビュータスクを実施
 - スコープ, 分岐条件, 引数, 関数呼び出し位置

結果

Task	Subject	Mean Search Time (s)		Success Rate (%)	
		UNIXDIFF	SYNCIA	UNIXDIFF	SYNCIA
Scope	Karma	420	-122	0	50
	PM2	360	+60	60	0
	Popcorn	136	+77	100	100

UNIX DIFFと大きな差は見られなかった

CanSites	PM2	349	+14	29	42
	Popcorn	296	-141	100	100

考察

- ▶ プログラムの振る舞いが変わらない変更
 - 変数名の変更, 条件の記述の仕方

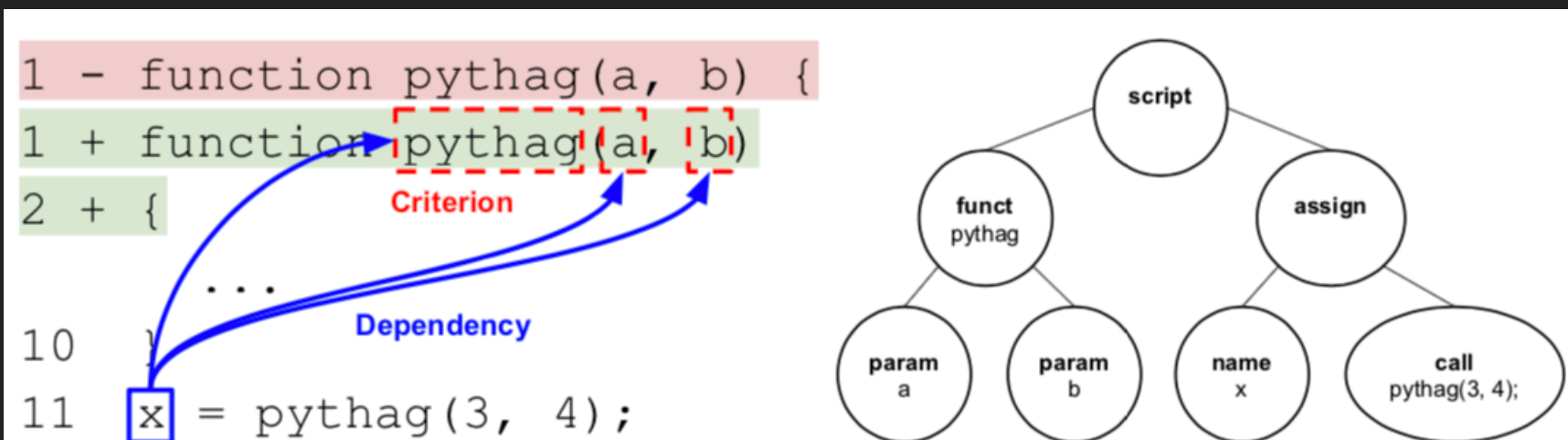


Fig. 2: Left – an example of a false dependency caused by a whitespace change and an imprecise diff utility. A change impact analysis might incorrectly infer that the value of `x` has changed based on the information provided by Unix diff that the definition of `pythag` has changed. Right – the AST diff for the same code (no change).

考察

▶ プログラムの動作が変わらない変更

変更前

```
1 function pythagorean(a, b) {  
2   var c;  
3   if(a == null || b == null)  
4     return undefined;  
5   a = Math.pow(a, 2);  
6   b = Math.pow(b, 2);  
7   c = Math.sqrt(a + b);  
8   return c;  
9 }  
10 pythagorean(3, 4);  
11 ;
```

変更後

```
1 function hypLength(a, b) {  
2   var c;  
3   if(!a || !b)  
4     return undefined;  
5   a = Math.pow(a, 2);  
6   b = Math.pow(b, 2);  
7   c = Math.sqrt(a + b);  
8   return c;  
9 }  
10 hypLength(3, 4);  
11 hypLength(6, 8);
```

```
1 - function pythag(a, b) {  
1 + function hypLen(a, b) {  
  ...  
9 }  
  ...  
11+ hypLen(6, 8);
```

関数名の変更

```
function hypLen(a, b) {  
  ...  
  if(a==null||b==null){  
    f(!a || !b) {  
      return undefined;  
    }  
  }  
  ...  
  hypLen(6, 8);
```

考察

▶ プログラムの動作が変わらない変更

変更前

```
1 function pythagorean(a, b) {  
2   var c;  
3   if(a == null || b == null)  
4     return undefined;  
5   a = Math.pow(a, 2);  
6   b = Math.pow(b, 2);  
7   c = Math.sqrt(a + b);  
8   return c;  
9 }  
10 pythagorean(3, 4);  
11 ;
```

変更後

```
1 function hypLength(a, b) {  
2   var c;  
3   if(!a || !b)  
4     return undefined;  
5   a = Math.pow(a, 2);  
6   b = Math.pow(b, 2);  
7   c = Math.sqrt(a + b);  
8   return c;  
9 }  
10 hypLength(3, 4);  
11 hypLength(6, 8);
```

```
1 - function pythag(a, b)  
1 + function hypLen(a, b)  
...  
9 }  
...  
11+ hypLen(6, 8);
```

条件の変更
関数呼び出し

```
1 + function hypLen(a, b) {  
...  
3 - if(a==null||b==null){  
3 + if(!a || !b) {  
4   return undefined;  
...  
11+ hypLen(6, 8);
```

考察

▶ プログラムの動作が変わらない変更

変更前

変更後

プログラムの動作に影響を及ぼさない
変更の抽出はコードレビューのノイズ

```
1 + function hypLen(a, b)
...
9 }
...
11+ hypLen(6, 8),
```

条件の変更
関数呼び出し

```
...
3 - if(a==null||b==null){
3 + if(!a || !b) {
4   return undefined;
...
11+ hypLen(6, 8),
```

- ▶ UNIX DIFFではなく AST DIFFを使用
 - 依存関係の捉えやすさ
- ▶ プログラムの比較とインターリーブ
- ▶ 構造差分を用いたインターリーブ
 - 自動化されており，高速

インターリーブ

- ▶ 同じプログラムポイントごとに、動作時の値を収容

PP	Statement in P_{old}	Values in P_{old}	Statement in P_{new}	Values in P_{new}
5	<code>a = pow(a, 2)</code>	<code>a : {9}</code>	<code>a = pow(a, 2)</code>	<code>a : {9, 36}</code>
6	<code>b = pow(b, 2)</code>	<code>b : {16}</code>	<code>b = pow(b, 2)</code>	<code>b : {16, 64}</code>
		<code>a : {9}</code>		<code>a : {9, 36}</code>
7	<code>c = sqrt(a+b)</code>	<code>b : {16}</code>	<code>c = sqrt(a+b)</code>	<code>b : {16, 64}</code>
		<code>c : {5}</code>		<code>c : {5, 10}</code>
8	<code>return c</code>	<code>c : {5}</code>	<code>return c</code>	<code>c : {5, 10}</code>



```
1 function pythagorean(a, b) {
2   var c;
3   if(a == null || b == null)
4     return undefined;
5   a = Math.pow(a, 2);
6   b = Math.pow(b, 2);
7   c = Math.sqrt(a + b);
8   return c;
9 }
10 pythagorean(3, 4);
11 ;

1 function hypLength(a, b) {
2   var c;
3   if(!a || !b)
4     return undefined;
5   a = Math.pow(a, 2);
6   b = Math.pow(b, 2);
7   c = Math.sqrt(a + b);
8   return c;
9 }
10 hypLength(3, 4);
11 hypLength(6, 8);
```

元プログラム

誤検知防止のための新定義

▶ 関数呼び出しの変更

	Interleaved Statement from P_{old}	Interleaved Statement from P_{new}	Concrete Call Stack P_{old}, P_{new}	Abstract Call Stack in P
1	funct foo() {	funct foo() {		
2	return;	return;	[7],[7]	[7:unchanged]
3	}	}		
4	funct bar() {	funct bar() {		
5	return;	return;	[],[8]	[8:changed]
6	}	}		
7	foo();	foo();	[],[]	[]
8	;	bar();	[],[]	[]

誤検知防止のための新定義

分岐条件の変更

	Interleaved Statement from P_{old}	Interleaved Statement from P_{new}	Concrete Condition P_{old}, P_{new}	Abstract Condition in P
1	if(x) {	if(x) {		
2	log('a');	log('a');	$[x], [x]$	$[x:\text{unchanged}]$
3	} else if(y) {	} else if(z) {		
4	log('b');	log('b');	$[!x, y],$ $[!x, z]$	$[!x:\text{unchanged},$ $y:\text{changed}]$
5	}	}		
5	return;	return;	$[], []$	$[]$

誤検知防止のための新定義

▶ 値の変更

	Interleaved Statement from P_{old}	Interleaved Statement from P_{new}	Concrete Store P_{old}, P_{new}	Abstract Store in P
1	$x = 1;$	$x = 1;$	$x : 1, x : 1$	$\{x : \text{unchanged}\}$
2	$y = 0;$	$y = 1;$	$y : 0, y : 1$	$\{y : \text{changed}\}$
3	$w = x;$	$w = x;$	$\{w : 1, x : 1\},$ $\{w : 1, x : 1\}$	$\{w : \text{unchanged},$ $x : \text{unchanged}\}$
4	$z = x * y;$	$z = x * y;$	$\{x : 1, y : 0,$ $z : 0\} \{x : 1,$ $y : 1, z : 1\}$	$\{x : \text{unchanged},$ $y : \text{changed},$ $z : \text{changed}\}$

誤検知防止のための新定義

▶ 変数の変更

	Interleaved Statement from P_{old}	Interleaved Statement from P_{new}	Concrete Environment P_{old}, P_{new}	Abstract Environment in P
1	<code>var x;</code>	<code>var x;</code>	x, x	$\{x, \text{unchanged}\}$
2	<code>var y;</code>	<code>var z;</code>	y, z	$\{z, \text{changed}\}$
3	<code>log(x);</code>	<code>log(x);</code>	x, x	$\{x, \text{unchanged}\}$
4	<code>log(y);</code>	<code>log(z);</code>	y, z	$\{z, \text{changed}\}$

SEMCIAの有用性評価のために

- ▶ UNIX DIFFに比べ、AST DIFFは有効か
- ▶ SYNCIAに比べ、SEMCIAは有効か
- ▶ SEMCIAは開発者にとって有用か

UNIX DIFF vs AST DIFF

TABLE VIII: Noise eliminated in the commit histories of MediacycenterJS, Karma and PM2. The second column group shows noise caused by using Unix diff as the source of structural change information. The third column group shows noise caused by using syntactic (control and data dependency) relations as a proxy for semantic relations.

Task	SEMCIA + AST diff		SEMCIA + Unix diff				Data & Control Dependencies				
	Criterion	Deps	Criterion	Increase	Deps	Increase	Criterion	Increase	Deps	Increase	
Variable	2,627	5,234	4,699	44%	10,190	49%	Data	29,968	1,041%	210%	
Value	14,414	17,910	26,409	29%	24,863	28%		108%	16,211	-9%	
Call	6,794	509	9,295	27%	696	27%	Control	24%	748%		
Condition	1,604	3,809	2,194	27%	4,932	23%		8,398	424%	13%	

抽出されたコード量
制御・データ依存関係が
23% ~ 49% 減少

SYNCIA vs SEMCIA

SYNCIA
SEMCIA

TABLE VIII: Noise eliminated in the commit histories of MediacenterJS, Karma and PM2. The second column group shows noise caused by using Unix diff as the source of structural change information. The third column group shows noise caused by using syntactic (control and data dependency) relations as a proxy for semantic relations.

Task	SEMCIA + AST diff		SEMCIA + Unix diff		Deps	Increase	Data & Control Dependencies			Deps	Increase
	Criterion	Deps	Criterion	Increase			Criterion	Increase			
Variable	2,627	5,234	4,699	44%	10,190	49%	Data	29,968	1,041%	16,211	210%
Value	14,414	17,910	26,409	29%	24,863	28%		108%	-9%		
Call	6,794	509	9,295	27%	696	27%	Control	24%	4,318	748%	
Condition	1,604	3,809	2,194	27%	4,932	23%		424%		13%	

抽出される依存関係は
SYNCIAに比べて少ない

SYNCIA vs SEMCIA

SYNCIA
SEMCIA

TABLE VIII: Noise eliminated in the commit histories of MediacenterJS, Karma and PM2. The second column group shows noise caused by using Unix diff as the source of structural change information. The third column group shows noise caused by using syntactic (control and data dependency) relations as a proxy for semantic relations.

不必要な情報が取り除かれたのでは？

ユーザスタディ

Task	Subject	Mean Search Time (s)			Success Rate (%)		
		UNIXDIFF	SYNCIA	SEMCIA	UNIXDIFF	SYNCIA	SEMCIA
Scope	Karma	420	-122	-192	0	50	80
	PM2	360	+60	-269	60	0	100
	Popcorn	136	+77	-80	100	100	100
Condition	Karma	147	-39	+1	75	100	100
	PM2	136	+77	-80	100	100	100
	Popcorn	90	+111	+10	100	100	100
Arguments	Karma	290	-24	-221	67	100	100
	PM2	284	-33	-239	50	67	100
	Popcorn	146	-48	-111	100	100	100
Callsites	Karma	272	-54	-80	100	100	100
	PM2	349	+14	-180	29	42	100
	Popcorn	296	-141	-174	100	100	100

従来の手法に比べ、ほとんどの項目で
有意な差が見られた

まとめ

- ▶ 従来の変更影響分析の課題を調査
- ▶ 課題解決のための新手法を提案
- ▶ 従来の手法に比べ、有効性、有用性があった

