JavaScript 網頁應用程式資料流測試模型之研究
A Study on Dataflow Testing Model of JavaScript Web Applications

劉建宏、陳偉凱、陳炳宏、林承甫
國立臺北科技大學資訊工程系
Chien-Hung Liu, Woei-Kae Chen, Ping-Hung Chen, Cheng-Fu Lin
Department of Computer Science and Information Engineering,
National Taipei University of Technology
Email :{cliu, wkchen, t100599001, t102598010}@ntut.edu.tw

摘　要

隨著 JavaScript 被廣泛應用於網頁前端應用程式的開發，其程式的品質日益受到大家的重視。相對於其他程式語言，JavaScript 具有不同的資料流特性，例如透過 DOM(Document Object Model) API 可存取 HTML 元件的屬性，並可藉由事件處理機制控管 HTML DOM 物件的觸發事件，達到與使用者互動的效果。為確保 JavaScript 程式對資料使用的正確性，本論文辨識並分析 JavaScript 的資料流資訊，進而提出一個 JavaScript 資料流測試模型，包含單一函數(Intra-procedural)、函數間(Inter-procedural)、單一頁面(Intra-page)與頁面間(Inter-page)的資料流程圖，以萃取和表示 JavaScript 網頁前端應用程式的資料流資訊。透過此模型，測試人員可以了解和分析 JavaScript 程式的資料流，協助其推導測試案例並分析可能的資料流異常情形。此外，本論文並實作一個輔助工具可以自動產生此模型，且計算出 JavaScript 變數之定義與使用的配對，並藉由實例展示此測試模型之有用性。

關鍵字：JavaScript 測試、資料流測試、網頁應用程式測試

一、前言

近年來，JavaScript 被廣泛應用於網頁前端程式邏輯處理，並隨著 HTML5 成為 W3C(World Wide Web Consortium)建議標準，JavaScript 在網頁前端開發的地位日益重要。因此，JavaScript 程式的品質亦逐漸受到大家的重視。

而相對於其他程式語言，JavaScript 具有獨特的資料流特性，例如能透過 HTML DOM 提供的介面存取與操作 HTML 文件中的所有元件，並擁有事件處理機制，以負責處理使用者或系統於 HTML DOM 物件觸發的事件，進而產生不同的資料流。而在不同頁面間，JavaScript 亦能透過本地端的儲存機制，如 HTML5 提出的 Web Storage，存取共同資料。

由於 JavaScript 程式語言的特性，開發人員可動態新增或修改程式中資料變數的資料定義與屬性，使得新增介面與修改屬性更加方便，有利於擴充功能；然而，這也可能造成資料定義改變而難以確認的問題。

目前已有相關研究提出方法或工具，能用以協助進行 JavaScript 的資料流分析(Data Flow Analysis)，大多數可達到單一函數(Intra-procedural)或函數間(Inter-procedural)的資料流分析，但少有涵蓋網頁中 HTML 元件與 DOM 事件處理的情形，以及本機端儲存資料的定義與使用，且未見到有提供 JavaScript 中資料變數的資料定義與使用關係。因此，希望藉由本研究得以從網頁前端 JavaScript 程式中，萃取出其資料變數定義與使用相關的資訊，進而了解其資料流程路徑，並用以協助測試人員找出涵蓋這些資料流路徑的測試路徑，確保設計之測試案例的有用性。

在後續的章節中，第二節將簡介資料流測試即其相關定義，並介紹 JavaScript 測試相關的研究與工具。第三節則將闡述本論文提出之測試模型的建構方法，以及如何計算出資料變數其定義與使用之配對。第四節總結本論文之研究，並提出未來研究方向。

二、相關研究

本論文採用資料流測試方法，取得 JavaScript 程式中變數定義與使用的資訊，分析其定義與使用配對的關係。

2.1 資料流測試

資料流測試強調資料的傳遞，其包含兩個基本的要素，定義(definition)與使用(use)。變數的定義為變數在何時被指派(assign)新的值，發生於變數的初始化、指派(assignment)，或接收參數時；變數的使用則表示變數在何時被參照、使用，而根據使用的方式不同可再分為兩種：c-use(computation use)和 p-use(predicate use)。前者為當變數使用於計算產生結果、作為其他變數定義的值或作為輸出的值的時候；後者則是當變數用於進行邏輯判斷，產生不同程式流程時。

當取得資料定義與使用的資訊後，可以找出各定義到其對應的使用位置之路徑，其間未有新的定義產生者稱為 definition-clear path，而此路徑之起點與終點－即其定義與使用之配對，則稱為 Def-Use pair，因此透過 Def-Use pair 可以瞭解到變數的定義與使用間的關係。在資料流測試中的應

用，若每個變數其所有的定義經 definition-clear path 到對應之使用，皆至少有一個測試案例覆蓋，便可滿足[1]中 All-Uses 的測試準則。

在本論文中，將以可抵達性定義分析(Reach Definition Analysis)[11]中的演算法計算取得變數定義的資訊，並觀察變數的使用情形，進而取得 Def-Use pair 進行分析。

2.2 JavaScript 分析與測試

在 JavaScript 分析的相關研究中，Jensen 等人於 2009 年所提出的[2]較具代表性。其提出一個靜態程式分析的架構(infrastructure)，透過使用抽象化的解譯(interpretation)推導 JavaScript 程式中詳細且完備的型別資訊，其分析的結果亦可用以顯示出一般的程式錯誤。此外，因[2]為針對 JavaScript 語言，並未涵蓋與網頁中 HTML 元件的互動，而於 2011 年提出的[3]再與以擴充，可對 JavaScript 程式與 HTML DOM 物件和流覽器 AP 的互動，如事件處理與 HTML DOM 物件的繼承架構，進行控制流程與資料的分析，其分析結果並能找出如使用消失的物件屬性，或辨識出無法抵達的(dead or unreachable)程式片段。

在 2014 年，Kashyap 等人於[4]提出一個 JavaScript 的靜態分析平台，為一基於 ECMAScript 3 中定義的抽象和實體的語法，抽象的直譯器，稱之為 JSAI。其計算包含資料型別的推論(type inference)、指標分析(pointer analysis)、控制流程分析(control-flow analysis)、字串分析(string analysis)與整數和布林常數的增值(integer and Boolean constant propagation)等的結果。

而在 JavaScript 網頁應用程式的測試方面，2011 年 Artzi 等人於[5]中提出一個 JavaScript 回饋應用的(feedback-directed)自動化產生測試的框架(framework)，可於執行時進行監測並收集相關的資訊顯示測試的涵蓋率，並將之實作為一工具，Artemis。2012 年 Mesbah 等人於[6]中提出一個方法，對基於 AJAX 技術的網頁應用程式進行動態分析，掃描 HTML DOM tree 並找出可能導致頁面狀態發生改變的元件，進而觸發事件爬取(crawling)其可能改變的網頁狀態，以及網頁狀態轉換時頁面瀏覽的路徑。

2.3 相關工具

WALA[7]為 IBM 所開發的開放原始碼工具，可提供 Java bytecode 和 JavaScript 的靜態分析，能進行函數間的資料流分析與指標分析並建立 call graph。另有一款開放原始碼的工具，dfatool[8]，能辨別 JavaScript 程式中變數的值與型別，甚至可應用於 JavaScript 模組(module)的使用，但亦不支援與網頁和 HTML DOM 的互動。而在[2]中，亦提供使用於 Eclipse 開發環境中的擴充套件[9]，可檢查一般的程式錯誤，如未被使用的變數、錯誤使用的函數等，亦可產生程式的 call graph 與解譯資料變數的定義。此外，[6]提供之工具，Crawljax[10]，能自動抓取網站所有可能的狀態，並以各狀態之網頁畫面縮圖呈現狀態圖，亦提供使用者設定對表單的自動輸入，以進一步抓取不同狀態。

在本論文中，將探討以 Reach Definition 方法應用於 JavaScript 資料流分析，為因應 JavaScript 資料流的特性，應如何修正演算法？並於第三節中提出建構模型之方法。

三、測試模型建構與資料流分析

本論文提出一個 JavaScript 資料流測試模型，可應用在網頁應用程式之測試。此模型包含 Intra-procedural、Inter-procedural、Intra-page，以及 Inter-page 資料流程圖，以表達 JavaScript 程式中單一函數內、不同函數間，以及單一頁面範圍內與不同頁面間的資料流的關係。以下章節將於 3.1 節分別介紹測試模型所包含的四個資料流程圖，並於 3.2 節描述其建構方法，最後再於 3.3 節中描述如何計算資料定義與使用的配對。

3.1 JavaScript 測試模型

為建構此模型，本論文探討 JavaScript 程式語言中特有的資料流，以 JavaScript 1.5 的版本為主，並於下列各子章節中分別闡述四種資料流程圖及其建構之方法。

3.1.1 Intra-procedural 資料流程圖

單一函數內可取得的資料定義可能來自於全域變數或區域變數，在 JavaScript 中，變數宣告於 Execution Context[12]，即程式的全域範圍以及執行的各函式主體，內層函式可取得所屬外層函式的資料定義，反之，則不成立。因此，在全域範圍所定義之變數即為全域變數；而各函式內定義之變數則為區域變數，僅可於該函式與其內層函式範圍內，取得該變數的定義。此外，在 JavaScript 程式中，當定義一未曾宣告過的變數，則此變數將自動成為全域變數。

透過 Intra-procedural 資料流程圖將呈現單一函數內的資料流，即資料變數在單一函數範圍內從定義到使用的情形。以圖 1 中的程式為例，變數 a 於第 1 列時被定義，在第 2 列中被用以進行邏輯判斷，而分別在第 3 與第 5 列重新定義。第 7 列進入 for 迴圈，使用 a 對變數 b 做初始化，當 b 的值小於 5 時，進入迴圈，並於第 8 列印出 b 的值至 console 視窗中，再於第 7 列更新 b 的定義。

```
1       var a = 0;
2       if (a > 0) {
3           a = 0;
4       } else {
5           a = 1;
6       }
7       for (var b = a; b < 5; ++b) {
8           console.log(b);
9       }
```

圖 1 Intra-procedural 資料流程圖範例程式

圖 2 則為此範例之 Intra-procedural 資料流程圖，其中，entry 節點(node)表示此函數範圍的進入點，表示函數流程由此開始，函數的參數便於此節點產生。exit 節點為函數之結束點，表示函數流程結束。而一般節點，以其對應之 JavaScript 表達式(statement)的所在列數表示(於本文敘述時以 N1 表示第 1 列的節點)，若有多個表達式存在同一列時，則再加上字母編號以做區別。每個連結(connection)的方向為資料流流向，當一節點為邏輯判斷，可產生 true 和 false 連結，表示根據判斷式結果產生之分支。

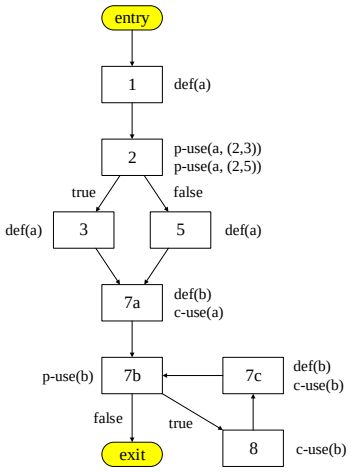

圖 2 Intra-procedural 資料流程圖示意圖

3.1.2 Inter-procedural 資料流程圖

Inter-procedural 資料流程圖主要表式函數間資料流的相關資訊，函數間資料流為當一變數 x 於一函數 A 中被定義，其資料於另一函數 B 中使用的情形。如圖 3，變數 a 於 N1 定義，而在 N10，呼叫使用定義的函數 foo。並在函數 foo 內，N3 與 N7 使用了變數 a，並在 N7 重新定義了 a。

其中 call node 表示函式呼叫的位置，call return 則表示其回傳位置，並以 call connection 與 return connection 函式呼叫與回傳的資料流方向。Inter-procedural 資料流程圖便可視為，透過 call、return 節點連結不同函式的 intra-procedural 流程圖，進而呈現不同函式間的資料定義與使用情形。

3.1.3 Intra-page 資料流程圖

Intra-page 資料流程圖為用以顯示出單一頁面中的資料流相關資訊，即當一資料變數其資料於函數 A 中定義，並於函數 B 中使用，且函數 A 與函數 B 並無直接呼叫關係。在網頁前端 JavaScript 可處理 DOM 物件觸發的事件，透過事件處理函式，可產生 Intra-page 資料流。

以圖 4 的程式為例，變數 a 與 b 在主要函數中被定義，而在函數 loadHandler 中，第 3 列有使用到 a，並於第 4 行使用 b 以重新定義 a，再重新定義 b。此外，在另一函數 clickHandler 中，於第 8 列，使用變數 a 作為 switch 判斷式，當 a 的值為 1 或 0 時皆於 console 畫面輸出 a 的值，且都使用 a 定義 b，並分別以遞減或遞增方式重新定義 a；當 a 的值不屬於以上兩者，便使用 b 去定義 a。最後分別註冊為 window 物件的載入事件與某一 HTML 物件的點擊事件，作為其事件處理器，而在程式中並未有直接呼叫兩者的情形。如此，單觸發事件後，便會產生 Intra-page 的資料流。

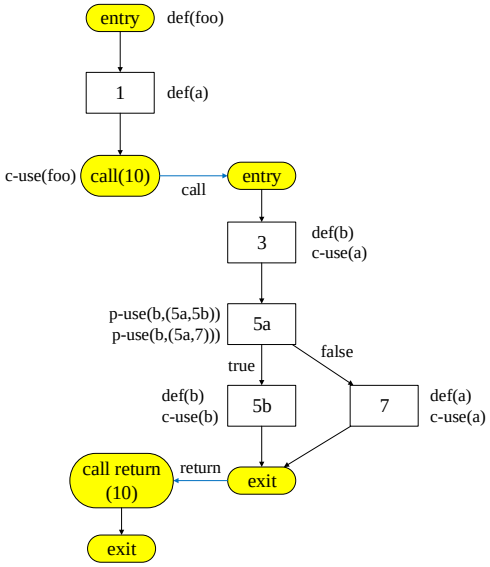

圖 3 Inter-procedural 資料流程圖示意圖

```
1       var a = 0, b = 1;
2       function loadHandler() {
3           if (a === 0) {
4               a = b--;
5           }
6       }
7       function clickHandler() {
8           switch(a) {
9           case 1: console.log(a);
10              b = a--;
11              break;
12          case 0: console.log(a);
13              b = a++;
14              break;
15          default: a = b;
16          }
17      }
18      window.addEventListener('load', loadHandler);
19      document.getElementById('btn')
            .addEventListener('click', clickHandler);
```

圖 4 Intra-page 資料流程圖範例程式

而在圖 5 中呈現此範例程式的 Intra-page 資料流程圖，其中，exception 連結表示在目前的節點可能產生 exception。而為模擬事件處理機制，loop node 用以表示程式於此等待事件觸發，並當事件觸發時透過建立 onEvent 連結進入至事件處理函式中，另外，loop return node 則是當事件處理器結束並回傳後皆透過 loop return node 再回到等待事件觸發的 loop node。

透過圖 5 可以瞭解到圖 4 的程式其 Intra-page 的資料流資訊，如在 loadHandler 函數中，於 N4

重新定義變數 a，當其回傳至 loop node，便可能再進入 clickHandler，而在 N9a、N9b、N10、N12a、N12b 和 N13 中使用。即變數 a 於同一頁面中未有直接呼叫關係的不同函數間，其資料定義與使用的關係。

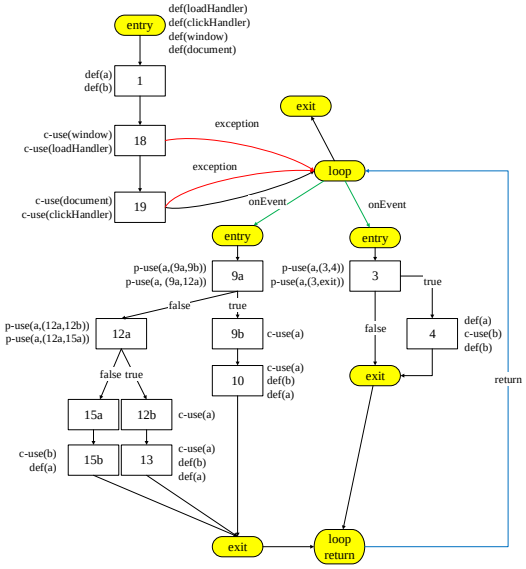

圖 5 Intra-page 資料流程圖示意圖

3.1.4 Inter-page 資料流程圖

而在網頁應用程式中，不同頁面間的資料交流，可以 local storage(於 HTML5 提出)儲存 key-value pair 資料進行。此類資料變數可供不同頁面間存取，進而可產生不同頁面間的資料流。

Inter-page 資料流程圖用以呈現在不同頁面間的資料流，以圖 6 的程式為例，localStorage 物件中 shared 屬性在頁面 a 中做為判斷式使用並可能重新產生定義，且在頁面 b 中亦使用到相同物件的屬性。

```
1    var sharedVar;
2    if (!!localStorage.shared) {
3        sharedVal = localStorage.shared;
4    } else {
5        localStorage.shared = 'new value';
6    }
```

圖 6 (a) Intra-page 資料流程範例頁面 a

```
1    var sharedVal = 'myVal';
2    var gotSharedVal;
3    if (!!localStorage.shared) {
4        gotSharedVal = localStorage.shared;
5    }
6    localStorage.shared = sharedVal;
```

圖 6 (b) Intra-page 資料流程範例頁面 b

若以 localStorage 物件的角度來看，當其屬性發生改變，亦代表 localStorage 產生新的定義，由此可得知，當先進入頁面 b 時，便於第 6 列定義 localStorage，而再導向至頁面 a 時，則會於第 2 列使用到 localStorage，並可於第 3 列再次使用，或於第 5 列產生新的定義，即產生頁面間的資料流。

在圖 7 的資料流程圖中，localStorage 節點用以表示 local storage 相關物件可於不同頁面中存取，即當頁面中有使用到 local stroage 相關物件時，便會產生一資料流從此節點至使用的位置；相對而言，當頁面中有對 local storage 相關物件重新定義時，便應有資料流從其產生定義的位置至 local storage 節點。

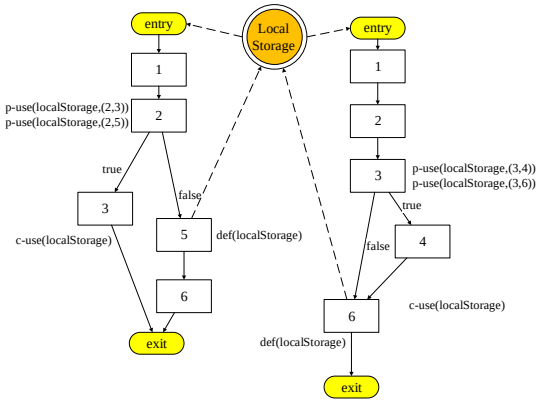

圖 7 Inter-page 資料流程圖示意圖

3.2 測試模型之建構方法

以下子章節將分別描述測試模型所包含四種資料流程圖：Intra-procedural、Inter-procedural、Intra-page 與 Inter-page 之建構方法。

3.2.1 Intra-procedural 資料流程圖

建置資料流程圖，需先產生目前範圍(scope)的控制流程圖(Control Flow Graph)，而在本論文中建構控制流程圖的方式是基於一開放原始碼(open source)工具，由 Arpad Borsos 所開發的 esgraph[13] 再進行延伸。其提供的方法所建置的控制流程圖為 Intra-procedural 的控制流程圖，故以此方式可直接產生 Intra-procedural 資料流程圖所需的控制流程圖。圖中的每一個節點皆對應於 JavaScript Parser 產生的 Abstract Syntax Tree 中的某一節點，並以類似 if-else 控制流程結構模擬 while、for 迴圈以及 switch 表達式的控制流程。

3.2.2 Inter-procedural 資料流程圖

在建構 Inter-procedural 資料流程圖前，需先知道呼叫函數(caller)與被呼叫函數(callee)的控制流程圖，且須能從程式中 callee 的名稱得知其函數範圍與控制流程圖，故須先完成 Intra-procedural 資料流程圖之建置，並取得其範圍內資料定義的相關資訊。而當兩函數範圍有呼叫關係時，再將兩函數之控制流程圖透過呼叫函數位置(call site)相連結，即當程式執行至呼叫函數呼叫被呼叫函數的位置，則建立一連結從 callsite 到 callee 的進入點，表示此程式控制流程從一函數範圍轉移至另一函數範圍內，並當 callee 結束而回傳時，應再建立連結由 callee 的結束點至 call site，用以表示 callee 執行流

程已結束，回到 call site 再繼續執行。然而當要計算變數的定義與使用時，若以同一節點表示函數呼叫與回傳，則會增加計算控制流程中接續的節點其資料定義的複雜度。因而在此所使用的方式為，分別以兩個節點，呼叫節點(call node)與呼叫回傳節點(call return node)，與 callee 的控制流程圖進行連結，建立呼叫與回傳的關係。圖 8 為建構 Inter-procedural 資料流程圖的演算法，圖 9 則為建構時連結呼叫與被呼叫函數的方法。

Algorithm: building Inter-procedural dataflow graph

Input: current scope
Output: Inter-procedural dataflow graph

```
1   get the current scope's CFG, G_curent
2   for each node n in G_curent do
3     if there is a call expression in n then
4       find the callee scope
5       get callee's CFG, G_callee
6       connect G_callee and G_curent as G_connect
7       recursive to the callee scope with G_connect
8       get updated CFG from recursive result as G_connect
9     end if
10  end for
11  return G_connect
```

圖 8 建構 Inter-procedural 資料流程圖演算法

Algorithm: connecting caller and callee's CFG

Input: CFG to connect, $G_{connect}$, callee's CFG, G_{callee} and the call site, N_{call}
Output: connected CFG

```
1   record nodes before N_call from G_connect to N_before
2   record nodes after N_call from G_connect to N_after
3   create call return node, N_return
4   connect N_return to all of N_call's descending nodes
5   disconnect N_call from its descending nodes
6   change the type of N_call to call node
7   connect N_call to G_callee's entry node
8   connect G_callee's exit node to N_return
9   if G_connect contains G_callee then
10    G_connect = N_before + N_call + N_return + N_after
11  else
12    G_connect = N_before + N_call + G_callee + N_return + N_after
13  end if
14  return G_connect
```

圖 9 連結 caller 與 callee CFG 之演算法

3.2.3 Intra-page 資料流程圖

當 JavaScript 程式中包含 HTML DOM 事件處理，且存在變數分別於不同事件處理器中被定義與使用，則將此情形視之為 Intra-page 資料流。為建構 HTML DOM 事件處理機制，本論文應用[14]建構類別中成員函式控制流程的方法於網頁前端的 JavaScript。在網頁前端，每個頁面為一個全域物件，視窗物件，而在此頁面定義之 JavaScript 函數與變數，皆為此視窗物件之成員，故應可適用[14]所提出之方法產生單一頁面之控制流程圖。而 HTML DOM 事件包含多種可由使用者或系統觸發之事件，且多數事件可重複觸發，除視窗載入事件，為載入網頁後最先觸發且僅此一次之事件，其後才等待其他事件的觸發。於控制流程圖中，以 loop node 表示程式於此將進行遞迴等待事件觸發，並接續所有事件註冊的處理函數，表示當所註冊之事件發生，則其控制流程便由此進入事件處理器中，當事件處理器回傳後，使其控制流程至 loop return node，表示事件處理完畢，系統將回復至等待狀態。

3.2.4 Inter-page 資料流程圖

網頁間交換資料的方式可以 cookie 或 web storage，將資料以鍵/值(key/value)的方式儲存於使用者端，供同一網域中不同頁面存取，從資料流的角度來看，可將 cookie 與 web storage 視為其定義可抵達頁面之物件，而在頁面中可存取其屬性，因而產生重新定義與使用此物件之情形。為建立頁面間資料流程的模型，需新增一位於各頁面範圍之外的節點，表示於此產生一變數與定義，可抵達各頁面，為表達此亦面間的資料流，以一個節點，本地儲存(Local Storage)表示。當有頁面中有使用 local storage 物件時，便有資料流自 local storage node 至該節點，且當頁面中有定義 local storage 物件時，亦有資料流從定義的位置至 local storage node。

3.3 資料定義與使用配對之計算

為取得 JavaScript 資料變數其定義與使用的配對，首先須能處理 lexical scoping 以取得資料變數可能的定義，而本論文中計算資料變數定義的方法將基於定義可達性分析(Reach Definition Analysis)[11]中的演算法如第(1)式：

$$\text{ReachOut(n)} = \text{ReachIn(n)} - \text{KILL(n)} + \text{GEN(n)} \quad (1)$$

以找出當程式執行至函數範圍對應的控制流程圖中各節點時可取得的變數與其資料定義。n 表示在控制流程圖上的任一節點，ReachIn(n)為可抵達至此節點的資料變數和定義的集合，GEN(n)為將會在此節點產生新的定義(如重新定義或變數宣告)的資料變數與定義的集合，而 KILL(n)則表示將會在此節點被重新定義的資料變數與其定義的集合，最後 ReachOut(n)便為可能抵達下一節點的資料變數與定義的集合。

3.3.1 Intra-procedural 資料定義計算

計算 Intra-procedural 資料流程圖中，變數定義的資訊，可直接使用第(1)式計算。而 JavaScript built-in 物件以及區域變數於進入 scope 時產生的 undefined 的定義，可視為於進入頁面程式主函式時產生。

3.3.2 Inter-procedural 資料定義計算

於 call node 時，並不會產生新的定義，因此可得第(2)式：

ReachIn(callee's entry)
= ReachOut(call node)
= ReachIn(call node) (2)

而在 call return node 時，需考量 call site 可能產生或刪除的定義，如第(3)式：

ReachOut(call return node) =
ReachIn(call return node)
– KILL(call site) + GEN(call site) (3)

且為了避免 callee scope 內的區域函數其定義可於外部存取，須將其從 callee 的 exit node 可取得的定義中刪除，如第(4)式：

ReachOut(callee's exit) =
ReachIn(callee's exit) – Def(local variables) (4)

此外，Inter-procedural 資料定義需考量到 caller 與 callee 之間的從屬關係，以圖 10 為例，program 範圍內定義了三個函式 fun1、fun2、fun3，fun1 則再包含了 funA。

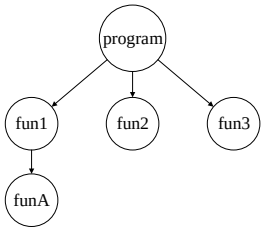

圖 10 scope tree

如當 funA 呼叫 fun2 的情形中，funA 可取得 fun1 內的資料定義，但是 fun2 不可以，因此當流程圖從 funA 進入 fun2 時，需扣除 fun1 中的定義以避免出現 fun1 的定義在 fun2 中使用的情形。

3.3.3 Intra-page 資料定義計算

而在計算單一頁面控制流程圖形中各點可抵達之定義時，因迴圈節點與迴圈回傳節點皆為抽象節點，不會產生新的資料定義，故可得第(5)與第(6)式：

ReachOut(Loop Node) = ReachIn(Loop Node) (5)
ReachOut(Loop Return Node) =
ReachIn(Loop Return Node) (6)

3.3.4 Inter-page 資料定義計算

對於各頁面共同使用之 local storage，因其定義可於不同頁面中使用，故需在各頁面的 entry node 加入 local storage 物件的定義，即第(7)式。

ReachIn(page's entry) =
ReachIn(page's entry) + Def(local storage) (7)

而當頁面中有定義 local storage 物件的情形時，便須將其定義加入至 local storage node 的 ReachIn，如第(8)式所示。

ReachIn(local storage) =
ReachIn(local storage) + {definition of local storage object generated from pages} (8)

四、結語

本論文提出一個 JavaScript 資料流測試模型，包含單一函數資料流程圖、函數間資料流程圖、單一頁面資料流程圖與頁面間資料流程圖，以進行資料流分析，並萃取網頁前端 JavaScript 程式中資料變數其定義與使用相關的資訊。透過此模型可幫助瞭解變數使用情形，並可用以輔助產生涵蓋所有變數定義與使用關係的測試路徑，以做為測試案例的參考。

未來藉由測試模型推導，並透過工具實作，以自動化產生能滿足資料流測試之測試準則(如 all-use 等)的測試路徑，並產生具良好可信度的測試案例，以能降低測試人員所需耗費的測試成本。此外，透過分析資料變數的定義與使用，進一步找出 JavaScript 網頁應用程式中資料流的異常情形。

致謝

本研究由科技部計畫 MOST 104-2221-E-027 -009 所補助，特此感謝。

參考文獻

[1] Mark New, "Data flow testing", Advance Topics in Computer Science, Swansea University, Wales, UK, 2002, 2012

[2] Simon Holm Jensen, Anders Møller and Peter Thiemann, "Type Analysis for JavaScript", Proceedings of the 16th International Symposium on Static Analysis, 2009, pp.238-255.

[3] Simon Holm Jensen, Magnus Madsen and Anders Møller, "Modeling the HTML DOM and Browser API in Static Analysis of JavaScript Web Applications", Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering, 2011, pp.59-69

[4] Vineeth Kashyap, Kyle Dewey, Ethan A. Kuefner, John Wagner, Kevin Gibbons, John Sarracino, Ben Wiedermann and Ben Hardekopf, "JSAI: a static analysis platform for JavaScript", Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2014, pp.121-132

[5] Shay Artzi, Julian Dolby, Simon Holm Jensen, Anders Møller and Frank Tip,"A framework for automated testing of javascript web applications", Proceedings of the 33rd International Conference on Software Engineering, 2011, pp.571-580

[6] Ali Mesbah, Arie van Deursen and Stefan Lenselink, “Crawling Ajax-Based Web Applications through Dynamic Analysis of User Interface State Changes”, ACM Transactions on the Web (TWEB), Volume 6 Issue 1, 2012, Article No.3
[7] WALA, http://wala.sourceforge.net/wiki/index.php/Main_Page
[8] dfatool - JavaScript Data Flow Analyze Tool, https://github.com/pissang/dfatool
[9] TAJS: Type Analyzer for JavaScript, http://www.brics.dk/TAJS/
[10] Crawljax - Crawling Ajax-based Web Applications, http://crawljax.com/
[11] Mauro Pezze and Michal Young, “Software Testing and Analysis: Process, Principles and Techniques”, United States of Americas: Jon Wiley & Sons, Inc., 2008, pp.82-84
[12] Dmitry Soshnikov, “ECMA-262-3 in detail. Chapter 2. Variable object”, http://dmitrysoshnikov.com/ecmascript/chapter-2-variable-object/
[13] Arpad Borsos, esgraph, https://github.com/Swatinem/esgraph
[14] Mary Jean Harrold and Gregg Rothermel, “Performing data flow testing on classes”, Proceedings of the 2nd ACM SIGSOFT symposium on Foundations of software engineering, 1994, pp.154-163W. S. Humphrey, *A discipline for software engineering*: Addison-Wesley Longman Publishing Co., Inc., 1995.