UNIVERSITÀ CA' FOSCARI - VENEZIA
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Specialistica in Informatica

Corso di Analisi e Verifica Programmi
Presentazione di progetto

Analisi Statiche sul Linguaggio JavaScript

Studente: Marco Lionello

Anno Accademico 2007-2008

Introduzione a JavaScript (1/4)

- JavaScript è
  - Un linguaggio di programmazione semplice
  - Permette l'interattività delle pagine web
- Breve storia
  - Nasce nel '95 da NetScape con il nome di LiveScript con lo scopo di allegerire il carico dai server
  - Microsoft crea Jscript e Vbscript
  - '97 ECMA(European Computer Manufactores Association) crea lo standard
  - Tuttora non esiste uno standard adottato al 100%
- Problemi di incompatibilità dei browser

Introduzione a JavaScript (2/4)

Javascript	Java
Interpretato	Compilato
Object-Based	Object-Oriented
Non è stand alone. Richiede ed è embedded in HTML.	È stand alone. Java è un ambiente di sviluppo completo.
Sviluppato da Netscape.	Sviluppato da Sun Microsystems.
Debolmente tipato	Fortemente tipato

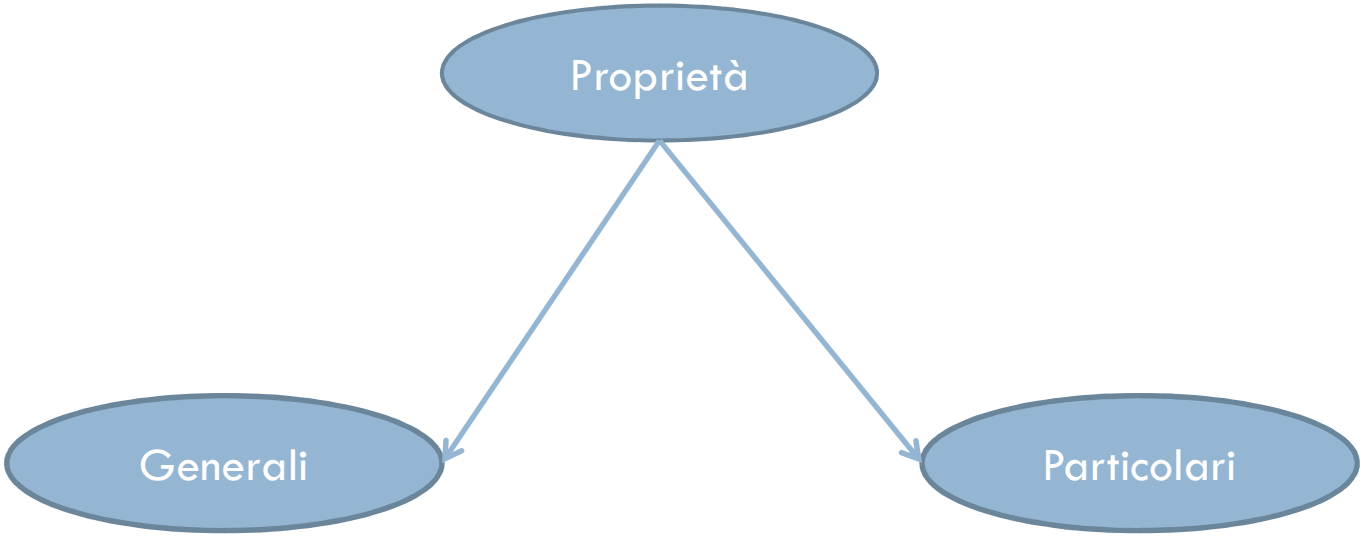
Introduzione a JavaScript (3/4)

Esempi di programmi	Esempi di programmi
Rollovers	Validazione di form
Menù a cascata	Moduli e Bottoni
Online Quiz	Gallerie e Slideshow
Convertitori	Date e orari
Calcolatrici	Finestre Pop-Up

Introduzione a JavaScript (4/4)

- Dichiarazione di varibili
  - Nel "C" si utilizzano solo variabili precedentemente dichiarate:
    - Ottimizzare l'uso della memoria
    - Aumentare l'affidabilità
  - In Javascript
    - Controllo di tipo lasco
      - Prova="testo" e Prova=59 sono valide
    - Se valori diversi vengono concatenati prevale String ma se la stringa è un numero sarà un numero
    - I risultati a volte sono imprevedibili es. moltiplicazione stringa numero
- JavaScript è un linguaggio object based
  - DOM
  - Ogni oggetto possiede
    - Un insieme di caratteristiche (proprietà)
    - Un insieme di cose che possono fare (metodi)
    - Un insieme di azioni di cui sono in attesa (eventi)

Proprietà dei programmi



Proprietà generali (1/2) Analisi Dataflow

- Assenza di comandi che assegnano un valore ad una variabile che non viene utilizzata prima di essere riassegnata
  - Dead code elimination → liveness analysis
- Assenza di variabili usate come costanti
  - Costant folding → reaching definition analysis
- Assenza di espressioni calcolate più di una volta
  - Common subexpression elimination → avaliable expression analysis
- Assenza di variabili che contengono sempre lo stesso valore
  - Copy propagation → reaching definition analysis
- Compatibilità dei browser
  - Combinazione di più analisi dataflow

Proprietà generali (2/2) Interpretazione Astratta

- Le proprietà che possono essere verificate tramite interpretazione astratta richiedono l'astrazione del dominio.

- Possono essere verificate proprietà
  - sul segno di espressioni aritmetiche
  - sul range di valori delle espressioni aritmetiche
  - sulla sicurezza
  - ecc.

- Si approfondirà solo quella relativa al segno delle espressioni aritmetiche

Proprietà particolari model checking

- Proprietà che dipendono strettamente da ciò che computa il programma

- Alcune proprietà riguardanti
  - Form di immissione dei dati
    - Tutti i dati devono avere un determinato formato
    - Tutti i campi obbligatori devono essere riempiti
  - Calcolatrice
    - Non deve essere permessa la divisione per 0
    - Ecc.

Analisi dataflow

- $gen_{LV}(p) = use[p]$
- $kill_{LV}(p) = def[n]$

$$LV_{exit}(p) = \begin{cases} \emptyset & \text{se p è un punto finale} \\ \bigcup \{ LV_{entry}(q) \mid q \text{ segue } p \} \end{cases}$$

$$LV_{entry}(p) = (LV_{exit}(p) \setminus kill_{LV}(p)) \cup gen_{LV}(p)$$

$$RD_{entry}(p) = \begin{cases} \iota & \text{se p è il punto iniziale} \\ \bigcup \{ RD_{exit}(q) \mid q \text{ precede } p \} \end{cases}$$

$$RD_{exit}(p) = (RD_{entry}(p) \setminus kill_{RD}(p)) \cup gen_{RD}(p)$$

$$AE_{entry}(p) = \begin{cases} \emptyset & \text{se p è il punto iniziale} \\ \bigcap \{ AE_{exit}(q) \mid (q,p) \text{ è un arco del grafo} \} \end{cases}$$

$$AE_{exit}(p) = (AE_{entry}(p) \setminus kill_{AE}(p)) \cup gen_{AE}(p)$$

Ottimizzazioni (1/3) Analisi dataflow

- Dead code elimination
  - Eliminazione del codice inutile
    - Istruzione del tipo s: a b c
    - Se a non è live out di s l'istruzione può essere eliminata
  - Possibili problemi con questa analisi nel caso di eccezioni (programma con debug vs programma originale)

- Costant propagation
  - Propagazione delle costanti
    - dichiarazione del tipo d: t c con c costante
    - dichiarazione di n: y t x con t usata da y
    - t è costante in n se d raggiunge n, e nessun'altra definizione di t raggiunge n
    - si riscrivere n come y c x

Ottimizzazioni (2/3) Analisi dataflow

- Common subexpression elimination
  - Data una dichiarazione del tipo s: t x y con l'espressione x y disponibile in s allora x y può essere eliminata
    - computa la reaching definition per trovare le reaching expression (trova le dichiarazioni nella forma n: v x y , tali che il percorso da n a s non computa x y o definisce x o y);
    - scegli un temporaneo w, e per n riscrivilo come:
      - n: w x y;
      - n:v w;
    - infine modifica la dichiarazione s s: t w .

Ottimizzazioni (3/3) Analisi dataflow

- Copy propagation
  - simile alla constant propagation, ma invece di avere una costante c abbiamo una variabile z.
  - Supponiamo di avere una dichiarazione d:t z e un'altra dichiarazione n che usa t cioè n:y t x.
    - Se d raggiunge n, e nessun'altra definizione di t raggiunge n, e non c'è nessun'altra definizione di z in qualsiasi percorso da d a n,
      - allora possiamo riscrivere n come n: y z x.

- Meglio usarla prima della common subexpression elimination
      - 1: a y + z
      - 2: u y
      - 3: c u + z
    - Le due espressioni di somma non vengono riconosciute come una subexpression elimination fino a che non viene efettuata una copy propagation di u a y

Incompatibilità del browser (1/5)

- Una chiamata a una proprietà o ad un metodo che non esiste lancia una eccezione del tipo
  - NoSuchFieldExcepion
  - NoSuchMethodException
- Ottenere la larghezza

```
function iw() {
  if (self.innerWidth) {
    return self.innerWidth;
  } else if (document.documentElement &&
          document.documentElement.clientWidth) {
    return document.documentElement.clientWidth;
  } else {
    return document.body.clientWidth;
  }}
```

Versioni di internet explorer antecedenti alla 6

Versioni di internet explorer 6 e 7

Altri browser

Incompatibilità del browser (2/5)

- Una selezione è un campo o un metodo richiesto in JavaScript
  - self.screenX
  - document.documentElement["clientWidth"]
- Incompatibility(n) indica l'insieme dei browser incompatibili con la selezione n
  - *Incompatibility(n)={x | x ∈ {browser} e l'esecuzione di n in x lancia un'eccezione}*
- Parametri del framework generale
  - E: punto iniziale
  - I: {IE6, IE7, Safari, ecc.}={browser}
  - F: nodi precedenti
  - Operatore insemistico: Intersezione
  - f: Lentry(n)\kill(n)

Incompatibilità del browser (3/5)

- Formalizzando abbiamo che:

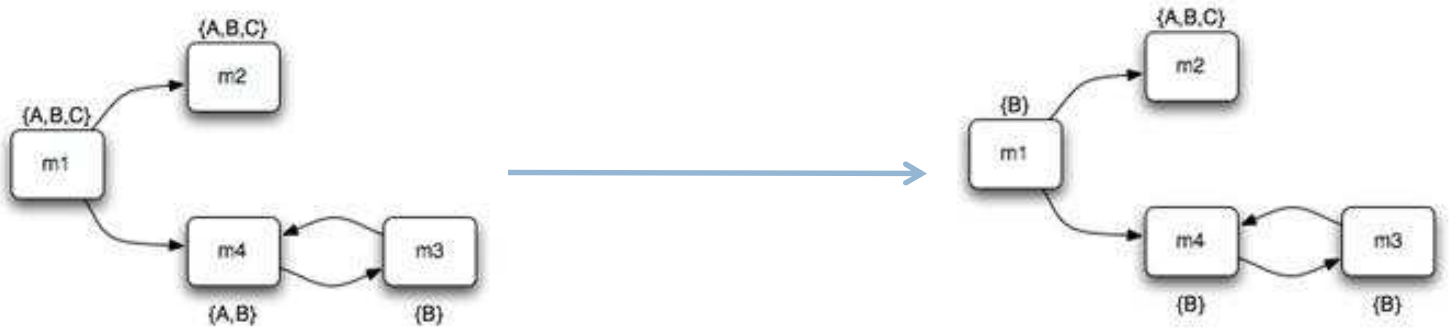
$$Gentry(n)=\begin{cases}\{Browser\} & se\ n\ nodo\ iniziale\\ \cap\{Gexit(q)|q\ precede\ n\} & altrimenti\end{cases}$$

$$Gexit(n)=Gentry(n)\backslash kill\ (n)$$

$$Kill(n)=\begin{cases}Incompatibility(n) & se\ n\ è\ una\ selezione\\ \emptyset & se\ n\ non\ è\ una\ selezione\end{cases}$$

  - In Gexit del nodo finale abbiamo l'insieme dei browser compatibili con la funzione p
    - Chiamiamo questa informazione *Incompatibility(p)*

Incompatibilità del browser (4/5)



- Parametri del framework generale
  - E: punto finale
  - I: {browser}
  - F: successore
  - Operatore insemistico: intersezione
  - f: G1exit(n)

Incompatibilità del browser (5/5)

- Formalizzando abbiamo che:

$$G1exit(n) = \begin{cases} \{Browser\} & se\ n\ nodo\ finale \\ \bigcap \{ G1entry(q) \mid q\ successore\ di\ n \} & altrimenti \end{cases}$$

$$G1entry(n) = G1exit(n) \cap supported(n)$$

  - In G1entry del nodo iniziale abbiamo l'insieme dei browser compatibili con il programma P

Interpretazione astratta (1/)

- Alcune caratteristiche di JavaScript rendono l'interpretazione astratta, basata sull'analisi statica, ineffettiva in particolare:
  - il carattere funzionale di JavaScript (in particolare l'abilità di ritornare un valore funzionale -implementato come chiusura- attraverso la funzione keyword), è difficile da analizzare attraverso l'interpretazione astratta;
  - il carattere riflessivo di javascript, che include la primitiva *eval* non è trattabile dalle tecniche di interpretazione astratta.
  - Le sue abilità di meta-programmazione non sono trattabili dalle tecniche di interpretazione astratta specialmente nei casi che questa meta-programmazione dipende dall'ambiente in cui opera.
    - Se si pensa che nel 46% delle pagine web contenenti JavaScript si fa uso di meta-programmazione, non implementare l'interpretazione astratta sulla meta-programmazione vorrebbe dire scartare il 46% dei codici da analizzare.
  - Il carattere prototipato ad oggetti di JavaScript è inusuale per la comunità che studia l'interpretazione astratta.

Interpretazione astratta (2/)

- La funzione eval eval("codestring");
  - se codestring è vuoto allora eval torna "undefined" ;
  - se codestring non è una stringa valida ritorna "undefined" ;
  - se la stinga è valida allora esegue il codice contenuto nella striga e ritorna il valore dell'ultima dichiarazione, se c'è una espressione valuta l'espressione e ritorna il valore appena valutato.

- Esempio

```
eval("fred=999; wilma=777; document.write(fred + wilma);");
```

Output:
1776

Interpretazione astratta (3/)

- Semantica concreta moltiplicazione

$$e = i \mid e^{*}e \mid eval(e)$$

$$\mu: Exp \cup \{Undef\} \rightarrow Int \cup \{Undef\}$$

$$\mu(ei) = \begin{cases} ei \; se \; ei \neq Undef \wedge ei \notin String \\ Undef \; se \; ei = Undef \\ Undef \; se \; ei \in String \end{cases}$$

$$\mu(e1 * e2) = \begin{cases} \mu(e1) * \mu(e2) se \; e1, e2 \neq Undef \\ Undef \; se \; e1 \mid e2 = Undef \end{cases}$$

Interpretazione astratta (4/)

- Semantica astratta moltiplicazione

$\eta: Exp + \{Undef\} \rightarrow \{+, 0, -, \perp, T\}$

$$\eta(e1)=\begin{cases} + \; se \; e1 > 0 \\ 0 \; se \; e1 = 0 \\ - \; se \; e1 < 0 \\ Und \; se \; e1 = Undefined \\ String \; se \; e1 \in String \end{cases}$$

$\eta\,(e1 * e2) = \eta\,(e1)\;\&\;\eta\,(e2)$

&	+	0	-	Und	String
+	+	0	-	$\perp$	T
0	0	0	0	$\perp$	T
-	-	0	+	$\perp$	T
Und	$\perp$	$\perp$	$\perp$	$\perp$	T
String	T	T	T	T	T

Interpretazione astratta (4/)

- La funzione *eval*
  - Semantica concreta

$$\mu_{eval}(e1, .., en)=\begin{cases} Undefinded\ se\ e1, \ldots, en \notin Syntax \\ \square \\ \mu(en)\ se\ en\ ha\ operatore * \vee e1, \ldots, en \in Syntax \end{cases}$$

  - Semantica astratta

$$\eta: Exp + \{Undef\} \rightarrow \{+, 0, -, \perp, T\}$$

$$\eta_{eval}(e1, \ldots en)=\begin{cases} \perp se\ e1, \ldots, en \notin Syntax \\ \square \\ \eta(en)\ se\ en\ ha\ operatore * \vee e1, \ldots, en \in Syntax \end{cases}$$

  - Semantica astratta 2

$$\mu_{eval}(e1, .., en)=\begin{cases} \perp se\ e1, \ldots, en \notin Syntax \\ \\ \perp altrimenti \end{cases}$$

Interpretazione astratta (5/5)

- In definitiva
  - Utilizzando la prima semantica astratta bisogna valutare qualsiasi comando solo per valutare il segno di espressioni aritmetiche
  - Utilizzando la seconda semantica astratta si ha una perdita totale di informazione

- Esempio:

```
5 * eval("fred=999; wilma=777; document.write(fred + wilma);");
```

  - se prendiamo la prima semantica astratta di eval risulta che non è possibile applicare l'interpetazione astratta perchè document.write non è definito;
  - per la seconda semantica di eval il risultato è (Perchè + &).

Proprietà particolari model checking

- Validazione di una form
  - Campi
    - Nome (obbligatorio)
    - Indirizzo
    - E-mail (obbligatorio)
  - Eventi
    - N immissione del nome
    - I immissione dell'indirizzo
    - E imissione dell'e-mail
    - T validazione
  - Proprietà atomiche
    - Pn: è stata appena compilata la form Nome
    - Pi: è stata appena compilata la form Indirizzo
    - Pe: è stata appena compilata la form E-mail
    - Pok: la form è valida
    - Pv: le form sono vuote

Struttura di Kripke versione 1 model checking

- S = {N1 , I1 , E1 , N21 , N22 ,I21 , I22 , E21 , E22 , N3 , I3 , E3 , Si , Sf}
- E = {N,I,E,T}
- So = { Si }
- L = {N1→{Pn}, I1→{Pi} , E1→{Pe} , N21→{ Pn } , N22→{ Pn } ,I21→{ Pi } , I22→{ Pi } , E21→{ Pe } , E22→{ Pe } , N3→{ Pn } , I3→{ Pi } , E3→{ Pe } , Si→{Pv} , Sf→{Pok} }
- R= { (N1,N,N1) , (N1, I,I21), (N1, E , E21), (N1, T , Si), (I1,N,N21) , (I1, I, I1), (I1, E , E22), (I1, T , Si), (E1,N,N22) , (E1, I, I22), (E1, E , E1), (E1, T , Si), ... }

Proprietà di Kripke1 model checking

- Proviamo a vedere che ogni esecuzione che porta allo stato finale ha al suo interno almeno Pn e Pe.

$$Se\ Pok\ vale\ al\ tempo\ t:$$

$$allora\ \exists\ t^{I} < t, t^{II} < t: Pn\ vale\ in\ t^{I} \wedge Pn\ vale\ in\ t^{II}$$

- Una proprietà che invece non vale è la seguente: ogni esecuzione che porta allo stato finale contiene almeno Pn e Pi.
  - Questa proprietà non vale e il model checker restituisce il contro esempio
    - Si → N1 → E21 → Sf

Automa sulle stringhe valide model checking

- Tre automi
  - E-mail controllo più sofisticato

- Automa e-mail
  - Si vuole che il campo e-mail oltre a contenere caratteri validi contenga anche il carattere @ che non sia in posizione finale ne iniziale
    - L rappresenta l'insieme {A , B,… , Z} ;
    - N rappresenta l'insieme dei caratteri {0 , 1 , 2 , … , 9};
    - A rappresenta l'insieme dei rimanenti caratteri.
  - Proprietà atomiche
    - *Pv* è la proprietà "la stinga è valida";
    - *Pi* è la proprietà "la stringa non è valida";
    - *P@* è la proprietà "è appena stata digitato @";
    - *Pl* è la proprietà "è stato appena digitato L o N";
    - *Pc* è la proprietà "il contatore è valido";
    - *Pe* è la proprietà "è stato digitato un carattere non valido o count invalido".

Struttura di Kripke versione 2 model checking

- S = { Si , V , I1 , I2 , I3};
- E = {N , A , L , @};
- So = { Si };
- L = { Si→{Pi}, V→{Pv} , I1→{Pi} , I2→{P@,Pi} , I3→{Pi} };
- R= { (Si,L,I1) , (Si,N,I3) , (Si,A,I3) , (I1,L,I1 if count<28) , (I1,L,I3 if count=28) , (I1,N,I1 if count<28) , (I1,N,I3 if count=28) , (I1,A,I3) , (I1,@,I2), (I2,@,I3) , (I2,A,I3), (I2,L,V) , (I2,N,V) , (V,L,V if count<30) , (V,L,I3 if count=30) , (V,N,V if count<30) , (V,N,I3 if count=30) , (V,A,I3)};

Proprietà di Kripke2 model checking

- Se la stringa è valida allora la stringa contiene una digitazione di L o N seguita da una del carattere @ e infine una digitazione di L o N. La sua formalizzazione è la seguente:

$$\forall t, \forall n \text{ se vale } Pv \text{ al tempo } t \text{ allora:}$$

$$\exists\, t^I, t^{II}, / t^I < t^{II} \wedge t^{II} < t \wedge Pl \text{ vale in } t^I \wedge P@ \text{ vale in } t^{II} \wedge Pl \text{ vale in } t$$

- Se la stringa è valida allora il contatore in tutti gli stati precedenti è valido:

$$\forall t, \forall n \text{ se vale } Pc \text{ al tempo } t \text{ allora:}$$
$$\forall t^I \ / t^I < t \, Pc \text{ vale in } t^I$$

Kripke finale model checking

- Data la prima struttura e la seconda struttura si può definire la fusione tra esse.
- Possiamo definire regole temporali definite in CLT (pag48)
  - Safety

$$CTL: AG\,\neg(Pok \wedge ((\neg Pn \wedge Pe) \vee (\neg Pe \wedge Pn) \vee (\neg Pe \wedge \neg Pn)))$$

    - che assicura che in ogni cammino sempre non può valere Pok se non sono presenti sia Pn sia Pe.
  - Fairness

$$CTL: AF\,\neg(Pi \wedge Pv)$$

    - che assicura che in ogni cammino sempre non può valere Pok se non sono presenti sia Pn sia Pe.
  - Liveness

$$CTL: AG((Pe \wedge Pn)\ U\ Pok)$$

Bibliografia

- **Compiler Design Implementation** [Libro] / aut. Mucknick Steven. - [s.l.] : Morgan Kaufmann Publ, 1997.
- **Corso di Analisi e Verifica Programmi** [Online] / aut. Tino Cortesi. - 01 08 2007. - http://www.dsi.unive.it/~avp.
- **Introduction to JavaScript** [Rapporto] / aut. Fletcher Kathy. - [s.l.] : West Virginia University, 2002.
- **Model Checking Tutorial** [Rapporto] / aut. Mazzoni Paolo. - [s.l.] : Politecnico di Milano, 2006.
- **Modern Compiler Implementation in Java** [Libro] / aut. Appel A. W.. - [s.l.] : Cambridge Univ. Press, 1998.
- **Pascal-Louis Perez, Josh Wiseman CS343** [Online] / aut. Pascal-Louis Perez, Josh Wiseman. - Stanford University. - 11 10 2007. - http://cs343-spr0607.stanford.edu/index.php/Projects:Pascal-Louis_Perez,_Josh_Wiseman.
- **PICS and Javascript** [Online] // www.poesia-filter.org/. - 09 09 2007. - http://www.poesia-filter.org/pdf/Deliverable_5_1.pdf .
- **Principles of Static Analysis** [Libro] / aut. F.Nielson H.R.Nielson, C.Hankin. - [s.l.] : Springer Verlag, 1999.
- **Recipient Driven Correctness Framework for Mobile Code** [Rivista] / aut. Padmanabhan Krishnan James Larkin, Phil Stocks. - Australia : Bond University.
- **Systems and Software Verifications** [Libro] / aut. al. B.Berard et. - [s.l.] : Springer Verlag , 2001.
- **Tipi nei linguaggi di programmazione** [Online] / aut. Pietro Cenciarelli. - 15 7 2007. - http://www.dsi.uniroma1.it/~lpara/DISPENSE/sezione_tipi.pdf.