

Ingeniería del Software II

Pedro R. D'Argenio

Sobre esta materia

- En esta materia estudiaremos varias técnicas de análisis (automático o asistido) de software. Algunas de las técnicas también son aplicables al análisis de hardware. Estas técnicas involucran, entre otras cosas, la especificación, verificación y validación de sistemas de software/hardware.
- Principalmente, estudiaremos dos técnicas automáticas importantes, model checking y deducción semántica a través de SAT solving y técnicas formales para la derivación de test.
- El curso será teórico-práctico, y experimentaremos en el uso de herramientas (LTSA, Spin, Alloy, Uppaal, JTorX).

Burocráticas

- Horarios:

- Teóricos: Miércoles (Lab 30) y Jueves (Aula 12), 9:00–11:00
- Prácticos: Miércoles y Jueves (Lab 30), 11:00–13:00

- Moodle (<http://www.famaf.proed.unc.edu.ar/>):

- Se podrá encontrar todo el material de la materia
- Se notificarán los eventos importantes
- Se podrán realizar consultas

- La materia se evaluará según:

- **1 trabajo práctico obligatorio** en grupos de a 3 personas que involucra varias etapas.
- **2 evaluaciones parciales** sin recuperatorio.

Burocráticas

De acuerdo a ello se pueden conseguir alguno de los siguientes estados:

- **Promoción:** De acuerdo a lo establecido en la Ordenanza 4/2011, para obtener la promoción, el alumno deberá:
 - aprobar todas las evaluaciones parciales con una nota no menor a 6 (seis), y obteniendo un promedio no menor a 7 (siete),
 - aprobar todos los Trabajos Prácticos.
- **Regular:** De acuerdo a lo establecido en la Ordenanza 4/2011, para obtener la regularidad, el alumno deberá:
 - aprobar al menos el 60 % de los Trabajos Prácticos o de Laboratorio.

Burocráticas

De acuerdo a ello se pueden conseguir algunos estados:

Calificación final:
 $(P1+P2+TP)/3$

- **Promoción:** De acuerdo a lo establecido en la Ordenanza 4/2011, para obtener la promoción, el alumno deberá:
 - aprobar todas las evaluaciones parciales con una nota no menor a 6 (seis), y obteniendo un promedio no menor a 7 (siete);
 - aprobar todos los Trabajos Prácticos.
- **Regular:** De acuerdo a lo establecido en la Ordenanza 4/2011, para obtener la regularidad, el alumno deberá:
 - aprobar al menos el 60 % de los Trabajos Prácticos o de Laboratorio.

Aprobar el TP
con 6 o más

Charlemos fechas

Dom	Lun	Mar	Mie	Jue	Vie	Sab
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24

Propuesta Proyecto

Elección Herramienta

Definición Herramienta

1er Parcial

Informe Preliminar

Presentaciones

Revisiones

2do Parcial

Informe Final

Abril

Mayo

Junio

Bibliografía

Libros:

- Jeff Magee & Jeff Kramer. Concurrency: State Models & Java Programs (2nd edition). Wiley. 2006.
- Christel Baier and Joost-Pieter Katoen. Principles of Model Checking. The MIT Press. May 2008.
- Berard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P. Systems and Software Verification Model-Checking Techniques and Tools. Springer, 2001.
- Mordechai Ben-Ari. Principles of the Spin Model Checker. Springer, 2008.
- Daniel Jackson. Software Abstractions: Logic, Language, and Analysis. The MIT Press. 2006.
- Aaron R. Bradley and Zohar Manna. The Calculus of Computation: Decision Procedures with Applications to Verification. Springer, 2007.

Bibliografía

Artículos (lista incompleta):

- B. Alpern & F. Schneider. Defining Liveness. Information Processing Letter 21:181-185. 1985.
- B. Alpern & F. Schneider. Recognizing Safety and Liveness. Distributed Computing 2(3):117-126. 1987.
- Markus Müller-Olm, David Schmidt, y Bernhard Steffen. Model-Checking: A Tutorial Introduction. En A. Cortesi, G. Filé (Eds.): SAS99, LNCS 1694, pp. 330354. Springer-Verlag 1999.

El problema de la corrección del software

Poder garantizar la corrección del software que construimos es una tarea deseable. En algunas aplicaciones, es, sin duda, crucial:

- Software para equipamiento médico
- Software para el control de vehículos
- Software para el control de procesos
- (para algunos, algunas aplicaciones financieras)
- ...

Estos sistemas, cuyas fallas pueden ocasionar daños de gran importancia -- incluyendo la pérdidas de vidas humanas, catastrofes ecológicas, grandes pérdidas financieras, etc. -- se denominan **sistemas críticos**.

El problema de la corrección del software

Dado que los sistemas críticos que construimos es fundamentalmente responden a comportamientos reactivos, concurrentes y/o de tiempo real, las técnicas que veremos se aplican a una gama de sistemas mucho más amplia, incluyendo, por ejemplo, sistemas de servicios web.

- (para algunos, algunas aplicaciones)
- ...

Estos sistemas, cuyas consecuencias de fallo son de gran importancia -- incluyen catástrofes ecológicas, grandes pérdidas financieras, etc. -- se denominan **sistemas críticos**.

Las técnicas que estudiaremos son particularmente importantes para este tipo de sistemas.

Ingeniería del Software II

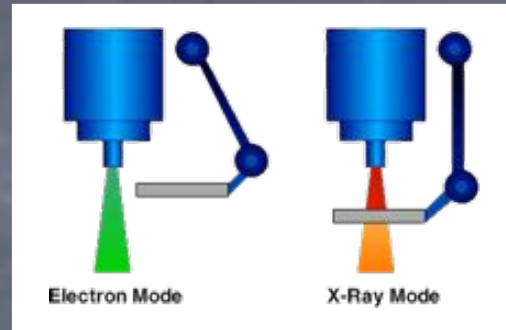
1 – Introducción

Algunos Ejemplos



Pentium:
FDIV

Ariane 5:
64 bits fp
vs 16 bits int



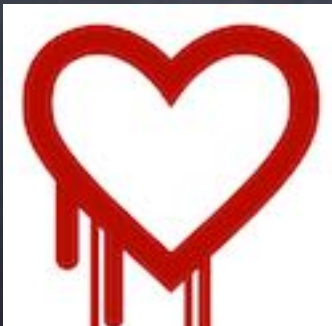
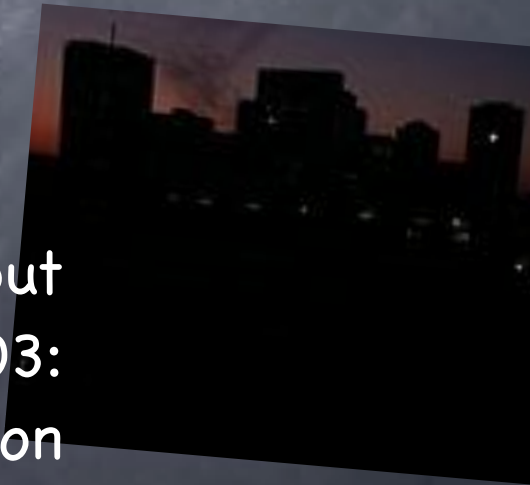
Therac-25:
Race condition



Mars Climate Orbiter:
Métrico vs Imperial



Northeast blackout
in 2003:
Race condition



Heartbleed:
Buffer over-read

Algunos Ejemplos



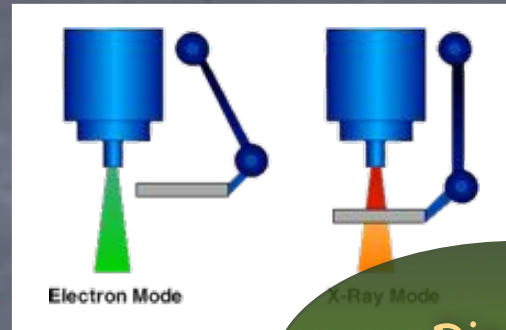
Pentium:
FDIV

$$\frac{4195835 * 3145727}{3145727} = 4195579$$

Ariane 5:
64 bits fp
vs 16 bits int



Integración



Therac-25:

Race condition

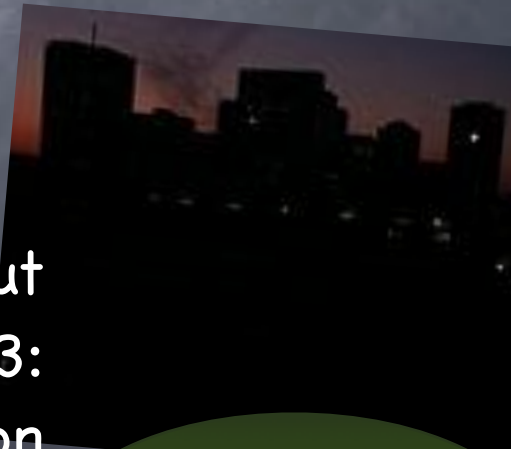
Diseño/
implementación

Mars Climate Orbiter:
Métrico vs Imperial



Integración

Northeast blackout
in 2003:
Race condition



Diseño/
implementación

Heartbleed:
Buffer over-read



Implementación

Algunos ejemplos

- Mars Pathfinder, 1997. Un problema de inversión de prioridades generaba que una tarea de alta prioridad se demorara. Un sistema de control entendía esta demora como un problema grave y producía un reset general.
- Y2K. Problema legado por los viejos programas que debían ahorrar memoria y se limitaban a guardar los dos últimos dígitos del año.
- Patriot Missile, 1991. 28 muertos y 100 heridos por un error de redondeo en el software de control de Patriot Missile.
- Airbus A320. Problemas en el software de control. El avión se estrella durante una demostración.
- DLR's, Londres: Los trenes paraban donde no se habían construido estaciones.

Algunos ejemplos

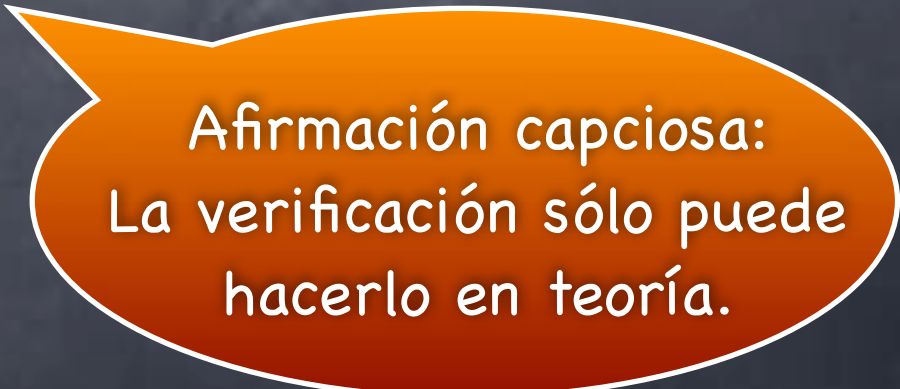
- Mars Pathfinder 1997. Un problema de inversión de prioridades

Muchos más bugs en:

- The risk digest: <http://catless.ncl.ac.uk/Risks/>
- Otros:
 - <http://www.cs.tau.ac.il/~nachumd/verify/horror.html>
 - <http://www5.in.tum.de/~huckle/bugse.html>
 - http://en.wikipedia.org/wiki/List_of_notable_software_bugs
- DLR's, Londres: Los trenes paraban donde no se habían construido estaciones.

Limitaciones del testing y la simulación

- Tanto el testing como la simulación involucran experimentos previos al lanzamiento o uso masivo del software. En general, ambos métodos proveen una serie de entradas al software, y estudian el comportamiento del mismo en esos casos.
- El testing y la simulación raramente permiten garantizar la ausencia de errores. Una frase famosa al respecto: "El testing puede confirmar la presencia de errores pero nunca garantizar su ausencia".



Afirmación capciosa:
La verificación sólo puede
hacerlo en teoría.

Verificación (semi)automática de software

- Existen serias limitaciones en lo que respecta a la verificación automática de software. Por ejemplo, el problema de decidir si un programa dado termina o no **no es computable**.
- Sin embargo, si imponemos algunas restricciones sobre las propiedades que queremos verificar, y sobre qué sistemas, algunas tareas pueden realizarse automáticamente. Model checking es un ejemplo de esto (que estudiaremos en más detalle más adelante).

Verificación Deductiva de Programas

La verificación de programas es la tarea de comprobar matemáticamente la corrección de un programa con respecto a su especificación. En el caso de programas secuenciales, existe desde hace varias décadas una técnica propuesta por Floyd y Hoare, basada en la especificación con aserciones en primer orden, y la visión de programas con aserciones como fórmulas de una lógica:

$$\left. \begin{array}{l} \{\text{Precondición}\} \\ \text{Programa} \\ \{\text{Postcondición}\} \end{array} \right\} \text{ Corrección total}$$

Esta “fórmula” indica que, bajo cualquier estado que satisfaga la precondición inicialmente, la ejecución del programa termina, y lo hace en un estado que satisface la postcondición.

Ejemplos

```
{ $x = X$  &  $y = Y$ }  
  aux := x;  
  x := y;  
  y := aux;  
{ $x = Y$  &  $y = X$ }
```

```
{true}  
  i := 1;  
  while (i ≤ length(A)) do  
    j := i;  
    while (j > 1) do  
      if (A[j-1] > A[j]) then  
        swap(j-1, j)  
      j := j-1  
    end  
    i := i+1  
  end  
{ $\forall x \in [1.. \text{length}(A) - 1] : A[x] \leq A[x + 1]$ }
```

Programas como aseercciones lógicas

Asignación

$$\frac{}{\{\psi[E/x]\} \quad x := E \quad \{\psi\}}$$

Composición en secuencia

$$\frac{\{\phi\} \quad p_1 \quad \{\eta\} \quad \{\eta\} \quad p_2 \quad \{\psi\}}{\{\phi\} \quad p_1; p_2 \quad \{\psi\}}$$

Ejecución condicional

$$\frac{\{\phi \wedge B\} \quad p_1 \quad \{\psi\} \quad \{\phi \wedge \neg B\} \quad p_2 \quad \{\psi\}}{\{\phi\} \quad \text{if } B \text{ then } p_1 \text{ else } p_2 \text{ fi } \quad \{\psi\}}$$

Iteración

$$\frac{\{\phi \wedge B\} \quad p \quad \{\phi\}}{\{\phi\} \quad \text{while } B \text{ do } p \text{ od } \quad \{\phi \wedge \neg B\}}$$

Consecuencia

$$\frac{\phi' \Rightarrow \phi \quad \{\phi\} \quad p \quad \{\psi\} \quad \psi \Rightarrow \psi'}{\{\phi'\} \quad p \quad \{\psi'\}}$$

Programas como aseercciones lógicas

Regla de inferencia para iteración:

La regla de inferencia para razonar sobre programas con iteración es lo que hace interesante al sistema de inferencia. Esta regla involucra nociones importantes, como las nociones de **invariante de ciclo** Inv y **función cota** t :

Si se cumple:

- (1) $\{\phi\} \text{ Init } \{Inv\}$
- (2) $\{C \wedge Inv\} \text{ CC } \{Inv\}$
- (3) $\neg C \wedge Inv \Rightarrow \psi$
- (4) $\{t = T\} \text{ CC } \{t < T\}$
- (5) $C \Rightarrow t > 0$

entonces:

```
{  $\phi$  }  
Init  
while  $C$  do  
    CC  
od  
{  $\psi$  }
```

Características de la verificación usando lógica de Hoare

- Puede extenderse a programas concurrentes (Owicki-Gries).
- No puede automatizarse (invariantes y funciones cotas son "mágicas").
- Introduce conceptos de gran importancia (aserciones, especificaciones, invariantes, etc.) que influyen en lenguajes y metodologías de programación.