

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 y Alloy, fundamentalmente).

Burocráticas

- ⌚ Horarios:
 - ⌚ Teóricos: Martes y Jueves, 9:00-11:00
 - ⌚ Prácticos: Martes y Jueves, 11:00-13:00
- ⌚ Acceso virtual:
 - ⌚ Aula Virtual: <https://famaf.aulavirtual.unc.edu.ar/>
 - ⌚ Zulip: #ingsoftii
- ⌚ 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

● Horarios:

- Teóricos: Martes y Jueves, 9:00-11:00
- Prácticos: Martes y Jueves, 11:00-13:00

- ❖ Todo el material de la materia
- ❖ Anuncio de eventos importantes
- ❖ Consultas y foro
- ❖ Videos grabados durante ASPO (2020 y 2021)

● Acceso virtual:

- Aula Virtual: <https://famaf.aulavirtual.unc.edu.ar/>
- Zulip: #ingsoftii

- ❖ Anuncios
- ❖ Consultas
- ❖ Foro de discusión

● La materia se evaluará según:

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

IMPORTANTE

La dirección de email asociada al aula virtual debe ser una dirección **activa** y chequeada regularmente.

Toda información importante será comunicada a través de avisos del aula virtual, si es información general, y a esas direcciones de email si es información particular

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 5 (cinco), 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
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	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

Propuesta Proyecto

Elección Herramienta

Definición Herramienta

1er Parcial

Informe Preliminar

Presentaciones

Revisiones

2do Parcial

Informe Final

Marzo

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.
- ⌚ Daniel Jackson. Software Abstractions: Logic, Language, and Analysis. The MIT Press. 2006.
- ⌚ A. Biere, M. Heule, H. van Maaren, y T. Walsh (eds.). Handbook of Satisfiability. 2nd edition. IOS press, 2021.
- ⌚ 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 industriales
- ⦿ (para algunos, algunas aplicaciones financieras)



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

El problema de la corrección del software

Dado que los sistemas críticos 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.

industriales

• (para algunos, algunas aplicaciones)



Estos sistemas, cuya importancia -- incluyendo 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

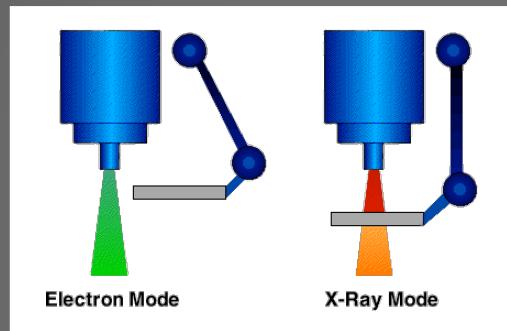
1 - Introducción

Algunos Ejemplos

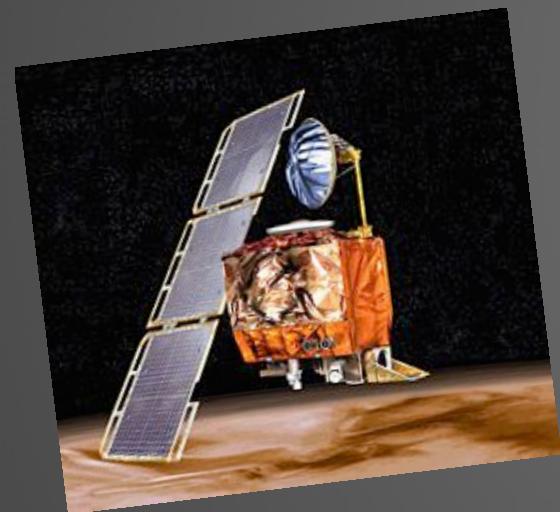


Pentium:
FDIV

Ariane 5:
64 bits fp
vs 16 bits int



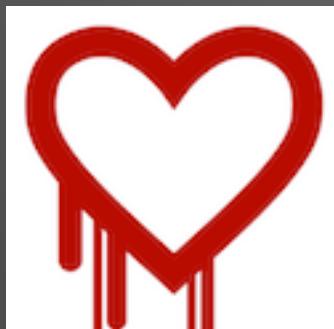
Therac-25:
Race condition



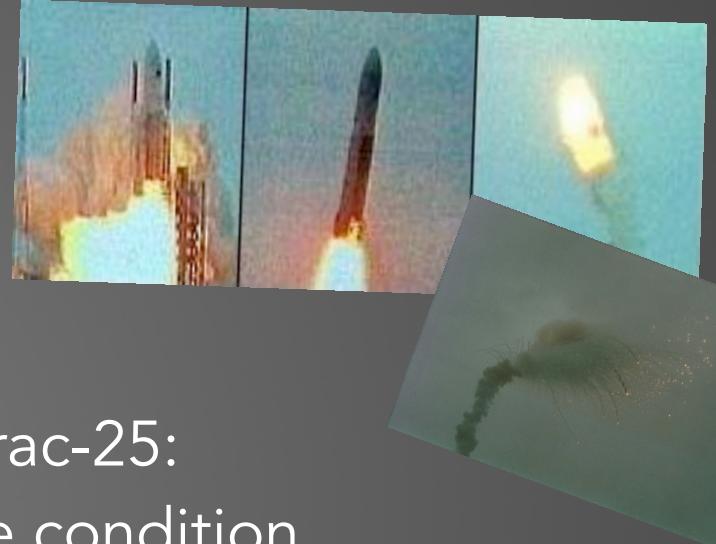
Mars Climate Orbiter:
Métrico vs Imperial

Northeast blackout

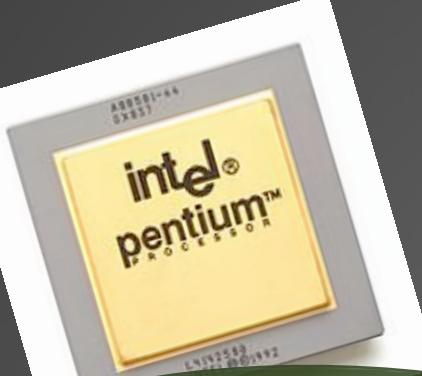
in 2003:
Race condition



Heartbleed
(Open SSL):
Buffer over-read



Algunos Ejemplos



Pentium:
FDIV

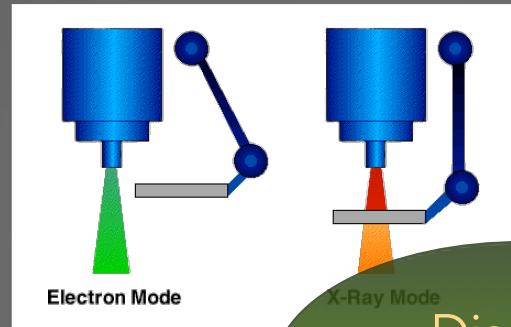
$$\frac{4195835 * 3145727}{3145727} = 4195579$$



Integración



Implementación

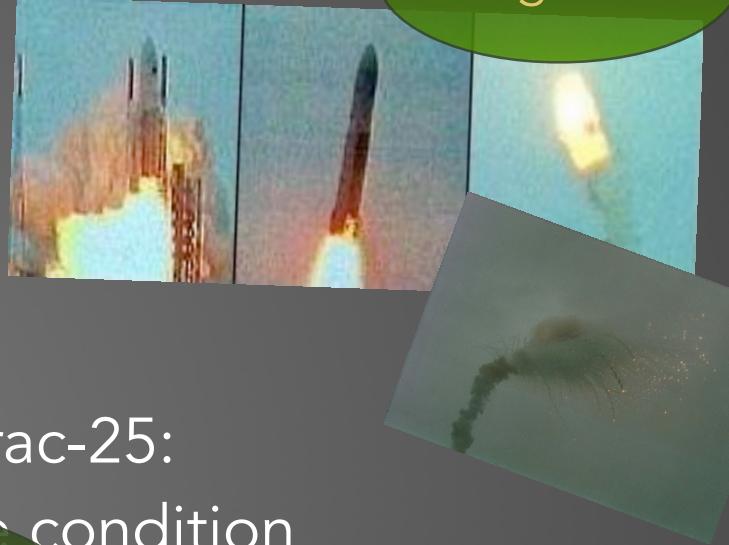


Mars Climate Orbiter:
Métrico vs Imperial

Therac-25:
Race condition

Diseño/
implementación

Ariane 5:
64 bits fp
vs 16 bits int



Northeast blackout
in 2003:

Race condition

Heartbleed
(Open SSL):
Buffer over-read

Diseño/
implementación

Algunos Ejemplos

Pentium:

Ariane 5:
64 bits fp

Integración

- ❖ 1985: <https://en.wikipedia.org/wiki/Therac-25>
- ❖ 1994: https://en.wikipedia.org/wiki/Pentium_FDIV_bug
- ❖ 1996: https://en.wikipedia.org/wiki/Ariane_5_Flight_501
- ❖ 1999: https://en.wikipedia.org/wiki/Mars_Climate_Orbiter
- ❖ 2003: https://en.wikipedia.org/wiki/Northeast_blackout_of_2003
- ❖ 2014: <https://en.wikipedia.org/wiki/Heartbleed>

open SSL

Buffer over-read

Implementación

Diseño/
implementación

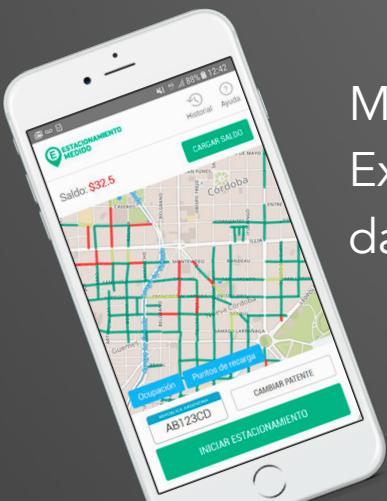
Algunos Ejemplos



911 blackout:
MAX value
reached



Nissan airbag:
Sensado
incorrecto



Movypark:
Exposición de
datos personales

Nest Thermostat:
Drenado de
batería



Boeing 737 MAX 8:
Sensado incorrecto

Tesla/Uber/Google
self-driving car:
aprendizaje con
limitaciones??



- ❖ <https://www.washingtonpost.com/news/the-switch/wp/2014/10/20/how-a-dumb-software-glitch-kept-6600-calls-from-getting-to-911/>
- ❖ <https://www.theverge.com/2014/10/20/7014705/coding-error-911-fcc-washington>
- ❖ <https://spectrum.ieee.org/nissan-recalls-nearly-1-million-cars-for-airbag-software-fix>
- ❖ <https://www.bbc.com/news/technology-35311447>
- ❖ <https://news.sophos.com/en-us/2016/01/18/nest-smart-thermostat-glitch-leaves-cold-feet-and-steaming-mad-customers/>
- ❖ <https://www.infobae.com/tecnologia/2019/10/22/una-falla-en-una-app-de-estacionamiento-medido-expuso-los-datos-de-miles-de-usuarios/>
- ❖ https://eldoce.tv/sociedad/exponen-datos-personales-traves-de-movypark-privacidad-error-falla-cordoba-empresa-estacionamiento-medido_91292/
- ❖ https://en.wikipedia.org/wiki/Boeing_737_MAX_groundings
- ❖ https://en.wikipedia.org/wiki/Self-driving_car#Incidents
- ❖ <https://www.craftlawfirm.com/autonomous-vehicle-accidents-2019-2024-crash-data/>
- ❖ <https://caraccidentattorney.com/blog/self-driving-car-accident-statistics/>
- ❖ <https://www.youtube.com/watch?v=E531GxfEoB8>
- ❖ <https://www.washingtonpost.com/technology/2023/06/10/tesla-autopilot-crashes-elon-musk/>
- ❖ <https://prospect.org/infrastructure/transportation/2023-06-14-elon-musk-tesla-self-driving-bloodbath/>
- ❖ <https://x.com/bdjukic/status/1740441354378051837>

Algunos Ejemplos



Schiaparelli enters atmosphere

Time: 0 sec
Altitude: 121 km
Speed: 21 000 km/h



Heatshield protection during atmospheric deceleration

Time of maximum heating: 1 m 12 sec
Altitude: 45 km
Speed: 19 000 km/h



Parachute deploys

Time: 3 m 21 sec
Altitude: 11 km
Speed: 1700 km/h



Front shield separates, radar turns on

Time: 4 m 1 sec
Altitude: 7 km
Speed: 320 km/h



Parachute jettisoned with rear cover

Time: 5 m 22 sec
Altitude: 1.2 km
Speed: 240 km/h



Thruster ignition

Time: 5 m 23 sec
Altitude: 1.1 km
Speed: 250 km/h



Thrusters off; freefall

Time: 5 min 52 sec
Altitude: 2 m
Speed: 4 km/h



Touchdown

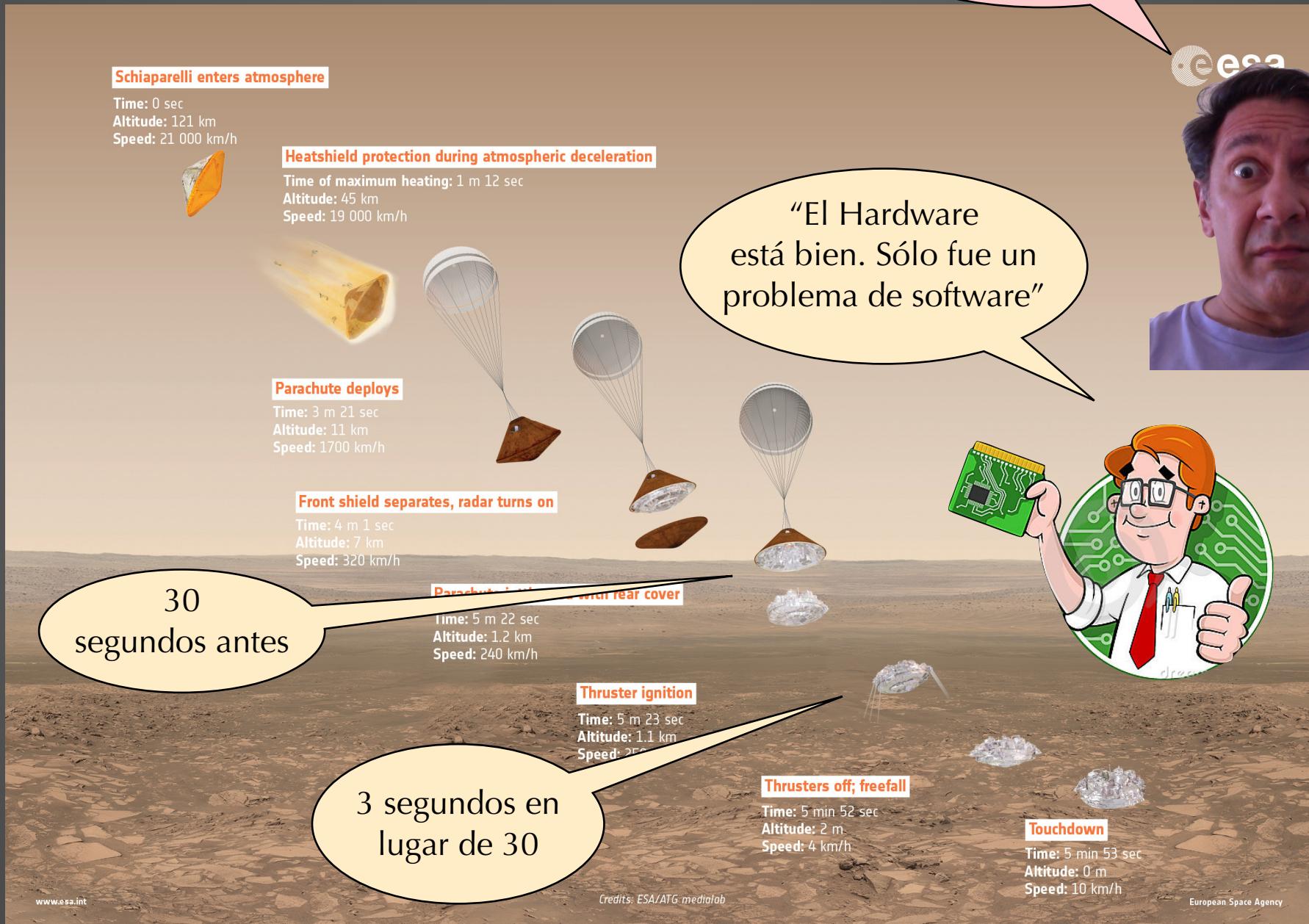
Time: 5 min 53 sec
Altitude: 0 m
Speed: 10 km/h



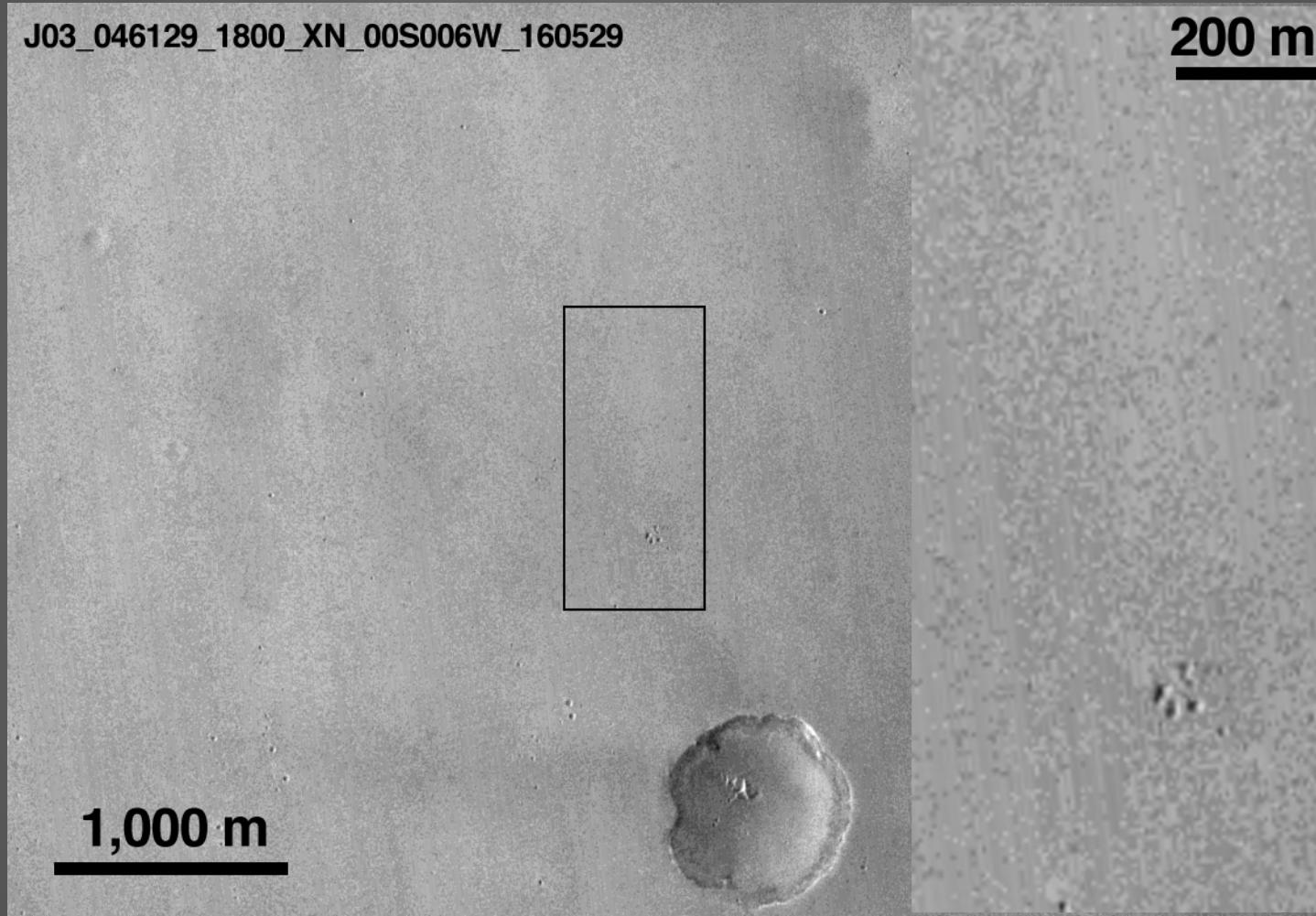
Credits: ESA/ATG medialab

Algunos Ejemplos

WTF!!!!

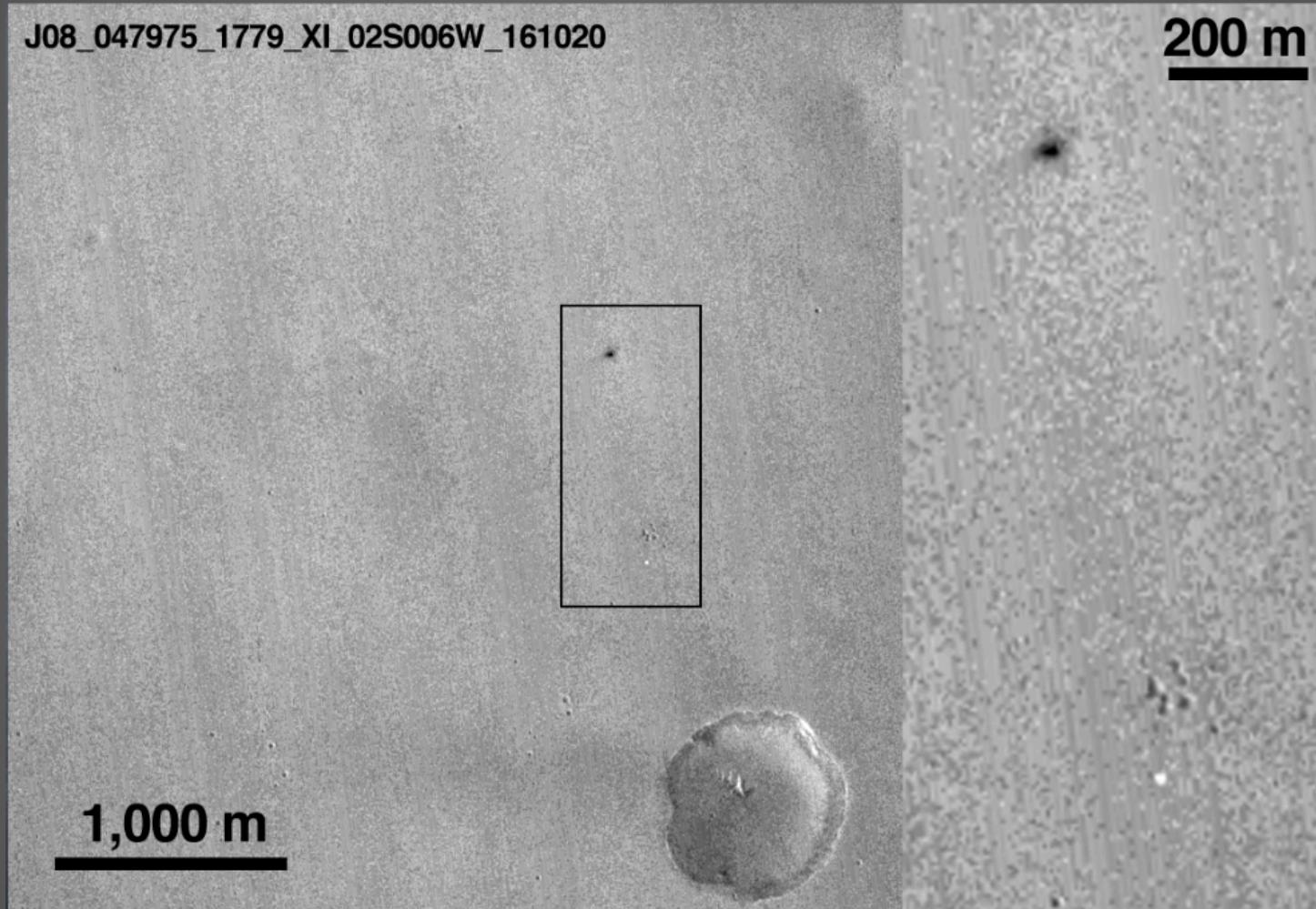


Algunos Ejemplos



https://en.wikipedia.org/wiki/Schiaparelli_EDM_lander

Algunos Ejemplos



https://en.wikipedia.org/wiki/Schiaparelli_EDM_lander

Algunos Ejemplos

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

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”.

Limitaciones del testing y la simulación

- ➊ 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.
- ➋ 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”.

Es importante saber que distintos métodos dan distintas garantías y es importante saber cuáles son.

Mmmhhh...
La verificación puede hacerlo pero en teoría

Verificación (semi)automática de software

- ➊ 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**.
- ➋ 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:

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

Esta “fórmula” indica que, bajo cualquier estado que satisfaga la precondition 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

$\{\forall x \in [1..length(A) - 1] : A[x] \leq A[x + 1]\}$

Programas como aserciones lógicas

Asignación

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

Composición en secuencia

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

Ejecución condicional

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

Iteración

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

Consecuencia

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

Programas como aserciones 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:

$$\begin{array}{l} \{\phi\} \\ \text{Init} \\ \text{while } C \text{ do} \\ \quad \text{CC} \\ \text{od} \\ \{\psi\} \end{array}$$

Características de la verificación usando lógica de Hoare

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