

# Métodos Formales para Ingeniería de Software

*Ma. Laura Cobo*

Departamento de Ciencias e Ingeniería de la Computación  
Universidad Nacional del Sur  
Argentina

Departamento de Ciencias e Ingeniería de la Computación – Universidad Nacional del Sur, Argentina

## Motivación

Los defectos de software causan **grandes** Fallas

**Pequeños** errores en sistemas técnicos pueden tener consecuencias catastróficas

Resulta particularmente cierto en **Sistemas tiempo real**. Algunos ejemplos:

- Controladores de envío de ambulancias
- Controles de dispositivos en plantas nucleares
- Controles de dispositivos espaciales
- Seguridad en sistemas de voto electrónico

## Motivación

Los defectos de software causan fallas **omnipresentes**

El software actualmente está en casi todas partes

Se encuentra en muchos dispositivos de la vida diaria:

- Teléfonos móviles
- Dispositivos inteligentes
- Tarjetas inteligentes (tarjeta para el transporte público)
- Automóviles (computadoras de abordo)
- Etc.

## Consiguiendo software fiable

La fiabilidad no es exclusiva del software, para el resto de las ingenierías también es de suma importancia

Así por ejemplo, en **ingeniería civil** algunas estrategias utilizadas para alcanzar la fiabilidad son:

- Cálculos y/o estimaciones **precisos** de fuerza, stress, etc.
- Redundancia de hardware (“hacer las estructuras un poco más fuertes de lo necesario”)
- Diseño robusto (una falla no es catastrófica)
- Separación clara de los subsistemas
- El diseño sigue patrones que se ha probado funcionan.

**estas estrategias .... ¿funcionan en la ingeniería de software?**

## Consiguiendo software fiable

### ¿por qué no funcionan?

- Los sistemas de software computan funciones **no continuas**. Algo tan simple como el cambio de un bit puede cambiar el comportamiento drásticamente.
- La redundancia, entendida como replicación, no protege contra los **bugs**.
- No hay una **separación** clara en subsistemas. Una falla local puede afectar a todo el sistema.
- El diseño de software tiene una **complejidad** lógica muy alta.
- Los ingenieros de software en general no están entrenados para enfrentar **correctitud**.
- Se favorece la eficiencia y el costo por sobre la **fiabilidad**.
- La práctica de diseño para software fiable está en un estado **inmaduro** particularmente para sistemas complejos, particularmente los distribuidos.



## Consiguiendo software fiable

¿ cómo garantizar correctitud ?

La estrategia principal .... **TESTING**

### Contra errores internos o bugs

Diseñar configuraciones de test  
representativas  
Controlar el comportamiento pretendido de  
los mismos

### Contra fallas externas

Inyección de fallas (memoria,  
comunicación)  
Propagación de fallas

Sin embargo el **testing** tiene limitaciones significativas ...

- Muestra la presencia de errores, no su ausencia.
- Representatividad de los casos de test. ¿cómo probar lo inesperado?  
¿Cómo probar los casos raros?
- Es una labor intensiva y por lo tanto .... costosa.

## Consiguiendo software fiable

### ¿ Qué son los métodos formales?

- Métodos rigurosos utilizados en el diseño y desarrollo de sistemas
- Utilizan matemática y lógica simbólica → **formalidad**
- Incrementan la confianza en un sistema
- Incluyen dos aspectos
- Hacen un modelo formal de ambos y utilizan **herramientas** para probar matemáticamente que la **ejecución formal del modelo** satisface los **requerimientos formales**.

## Consiguiendo software fiable

### ¿ Para qué métodos formales?

- Complementan otros análisis y métodos de diseño
- Son buenos en la detección de **bugs**, tanto en el código como en la especificación
- Reducen el tiempo de desarrollo, incluyendo **testing y mantenimiento**.
- Aseguran ciertas **propiedades** sobre el **modelo** del sistema
- Idealmente deben ser tan automáticos como sea posible.
- El entrenamiento en métodos formales incrementa la **calidad** del desarrollo



## Consiguiendo software fiable

Los **métodos formales** se pueden asociar con **testing**.

- Correr el sistema con las entradas elegidas y observar el comportamiento
  - **Elegidos al azar** puede encontrar bugs, pero no hay garantías
  - **Elegidos en forma inteligente** se realiza a mano, lo que lo hace costoso
  - **Elegidos en forma automática** requiere de una especificación formalizada
- Test de cubrimiento: otras entradas
- Test “oracle”: sobre observaciones

## Consiguiendo software fiable

La **especificación** es fundamental, indicar qué es lo que un sistema **debe** hacer

- Propiedades simples
  - **Safety** “cosas malas nunca sucederán” (exclusión mutua)
  - **Liveness** “lo bueno eventualmente sucederá”
- Propiedades generales de sistemas concurrentes o distribuidas: deadlock free, no starvation, fairness
- Propiedades no funcionales: ejecución, memoria, usabilidad
- Especificación de comportamiento completa:
  - El código satisface el contrato que describe su funcionalidad
  - Consistencia de datos, invariantes de sistema, modularidad, encapsulamiento
  - Equivalencia de programas
  - Relación de refinamiento

## Los métodos formales no se pensaron para:

- Mostrar la “correctitud” de sistemas completos.
- Reemplazar completamente al testing
  - Es importante recordar que los métodos formales trabajan sobre modelos, código fuente o a lo sumo bytecodes
  - Además hay muchas propiedades no formalizables.
- Reemplazar las buenas practicas de diseño

- No se puede verificar código confuso con especificaciones no claras
- Tampoco hay sistemas correctos sin requerimientos claros y buen diseño

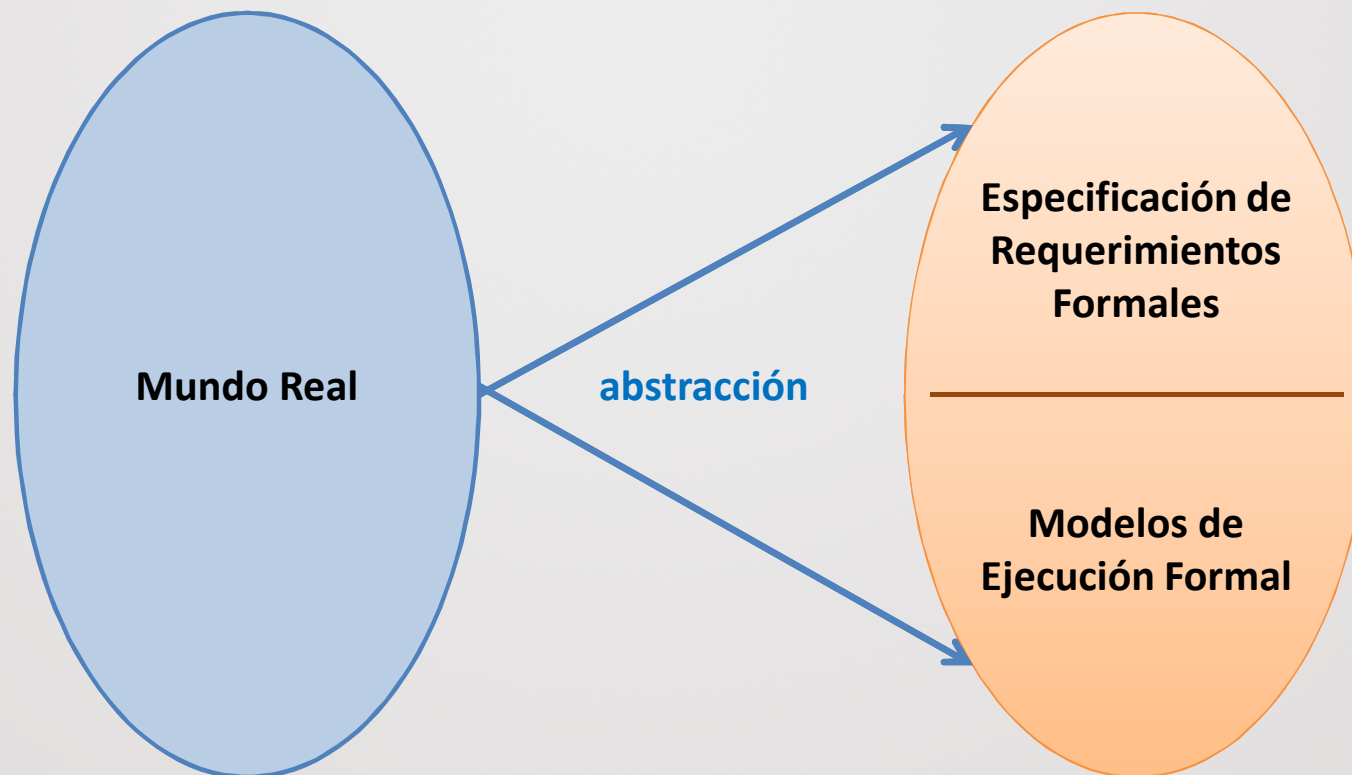
## Pero ...

- Las pruebas formales pueden reemplazar muchos casos de test.
- Los métodos formales mejoran la calidad de las especificaciones (aún cuando no se haga una verificación formal)
- Los métodos formales garantizan propiedades específicas del sistema modelado

**La formalización de los requerimientos de un sistema es difícil**

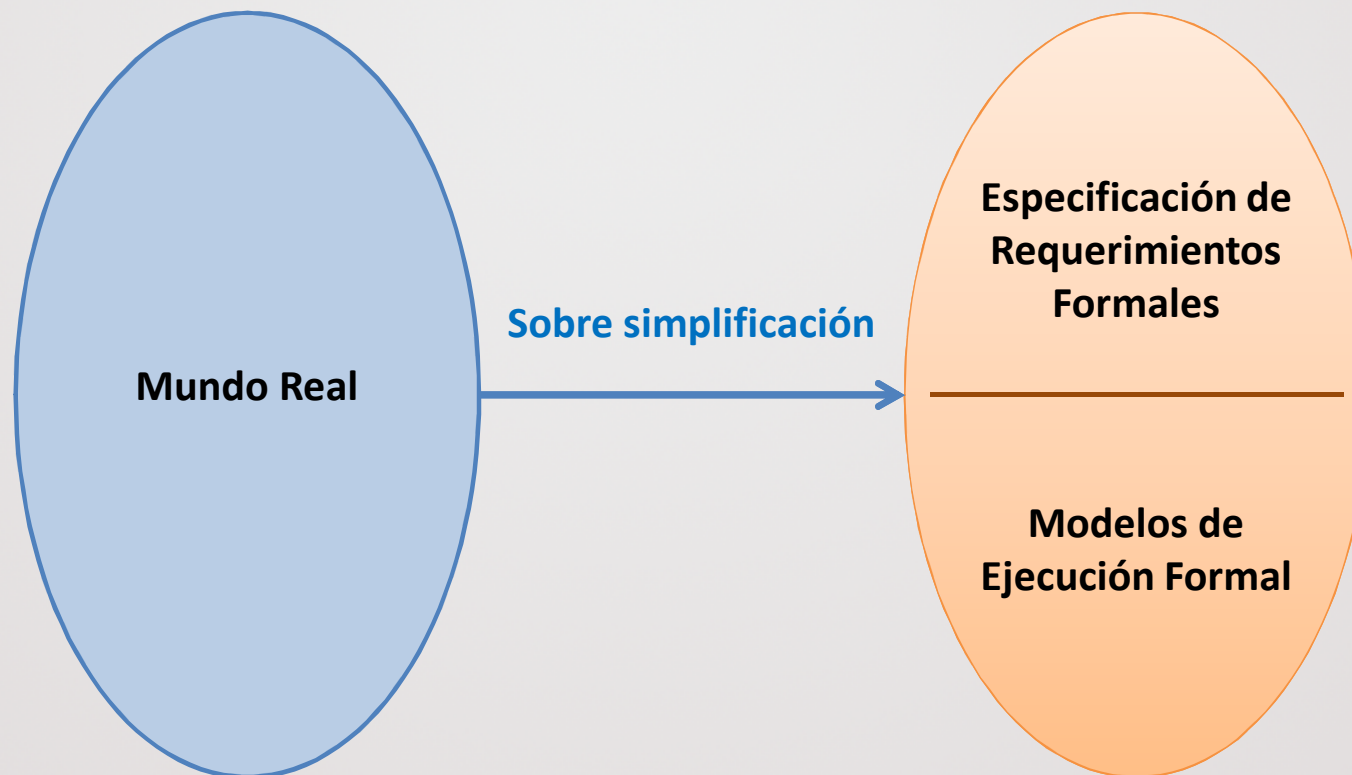
**Probar propiedades de un sistema puede ser difícil**

## Dificultades en crear modelos formales

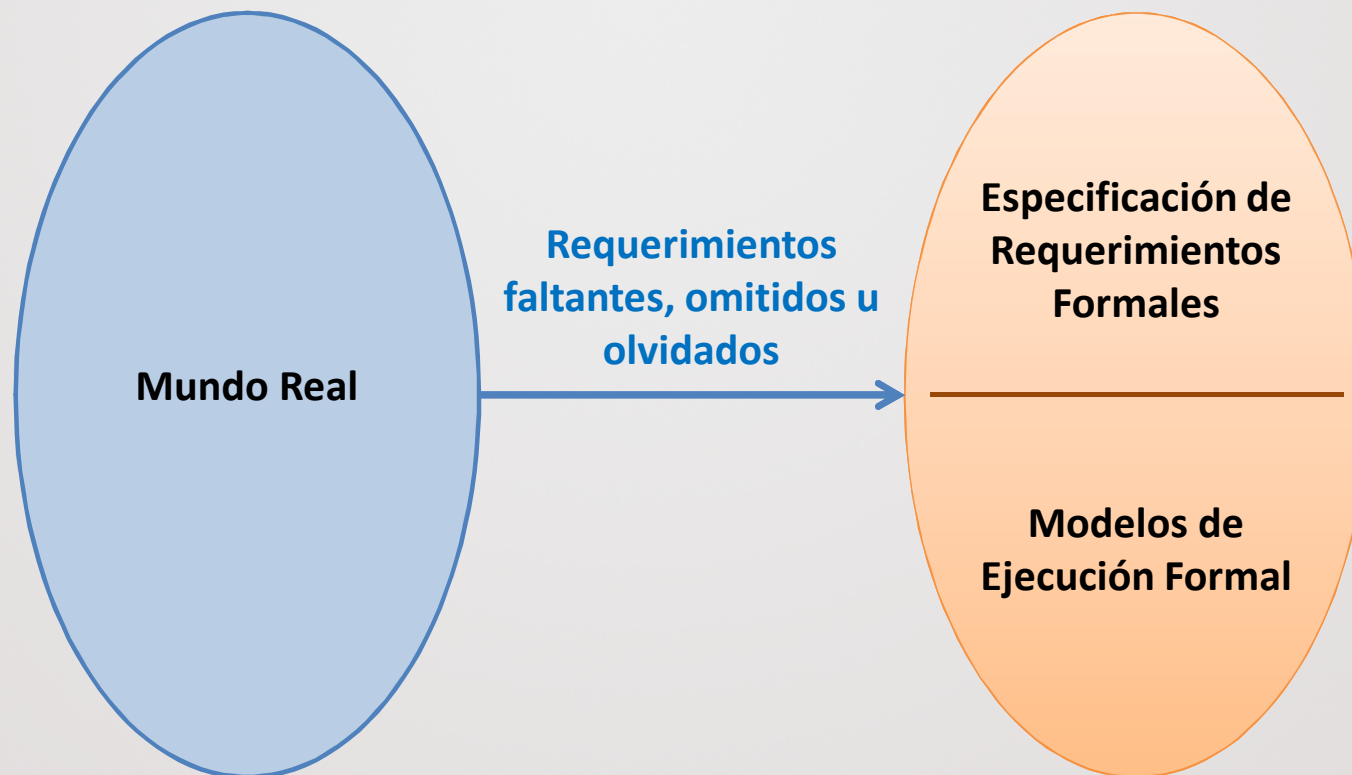




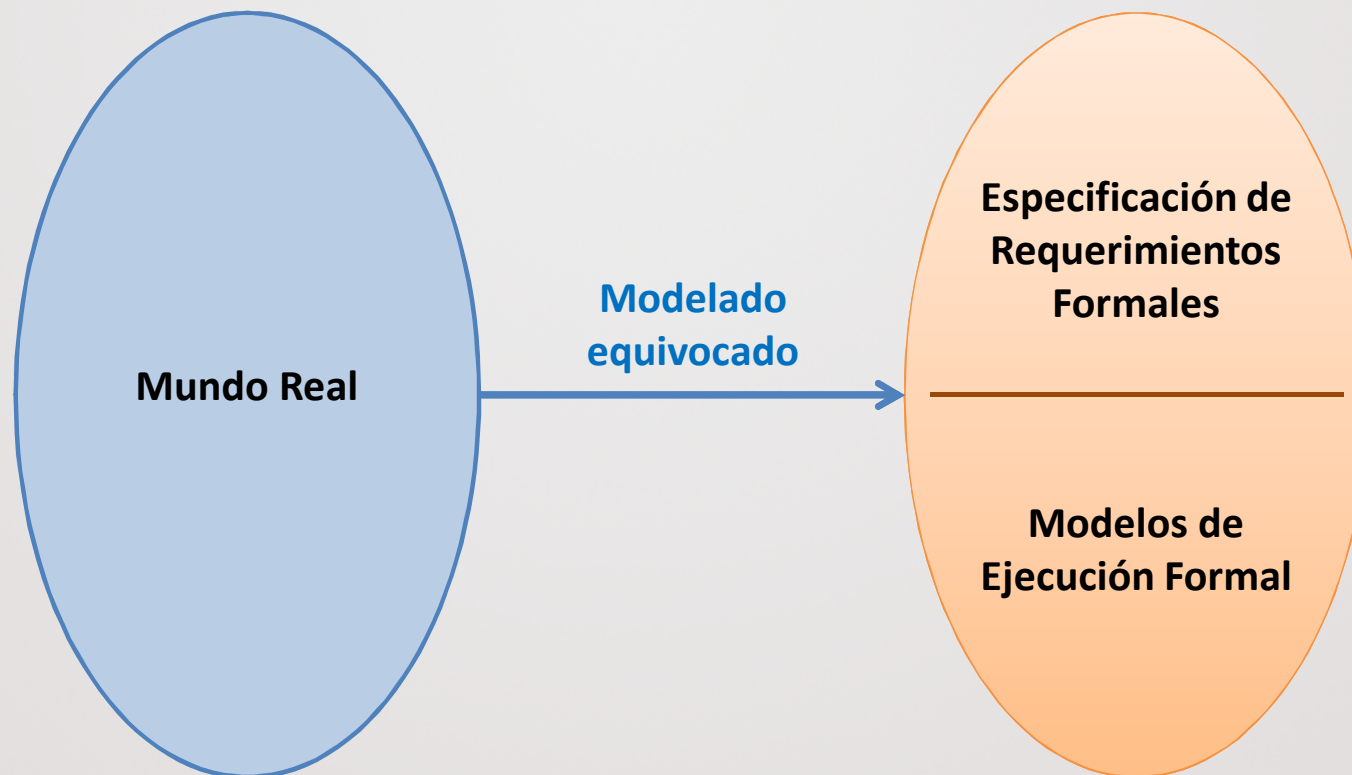
## Dificultades en crear modelos formales



## Dificultades en crear modelos formales



## Dificultades en crear modelos formales



Departamento de Ciencias e Ingeniería de la Computación – Universidad Nacional del Sur, Argentina

## Nivel de descripción del sistema

### Nivel abstracto

- Tipos finitos
- Es posible realizar pruebas automáticas
- Simplificación, modelos poco realistas

### Nivel concreto

- Tipos infinitos
- Datos complejos y estructura de control
- Modelo de programación real
- Las pruebas automáticas son en general inviables

## Expresividad de la especificación

### Simple

- Propiedades simples o generales
- Aproximación → baja precisión
- Es posible realizar pruebas automáticas

### Compleja

- Especificación de comportamiento completa
- Cuantificación sobre dominios infinitos
- Alta precisión → modelado más ajustado a la realidad
- Las pruebas automáticas son en general **inviabiles**



## Aproximaciones usuales

Programas abstractos  
Propiedades simples

Programas Abstractos  
Propiedades complejas

Programas Concretos  
Propiedades simples

Programas Concretos  
Propiedades Complejas

## Mecanismo de prueba

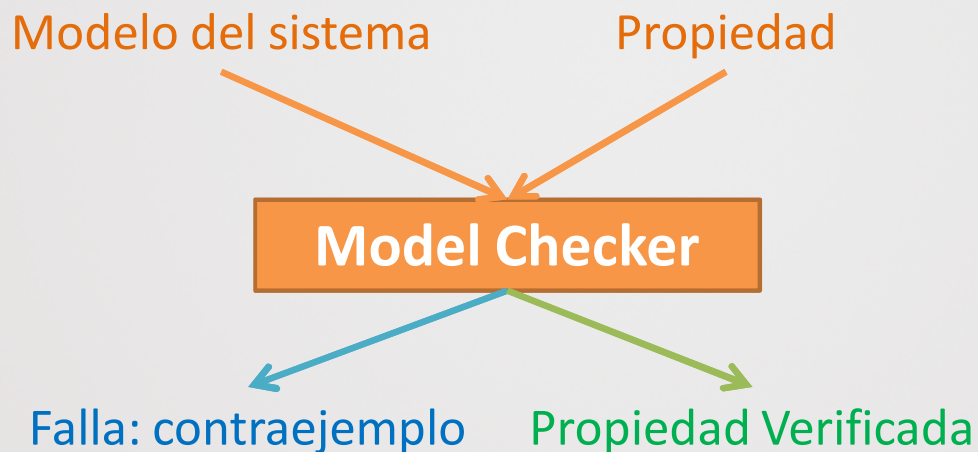
### Automático

- No se requiere interacción durante la verificación
- La prueba puede fallar o ser **no** conclusiva (requiriéndose una mejora en el seteo de los parámetros de la herramienta)
- Especificación formal realizada a “mano”

### Semi-automático

- Puede requerirse interacción durante la verificación
- **Se necesita conocimiento interno de la herramienta**
- La prueba es chequeada por la herramienta

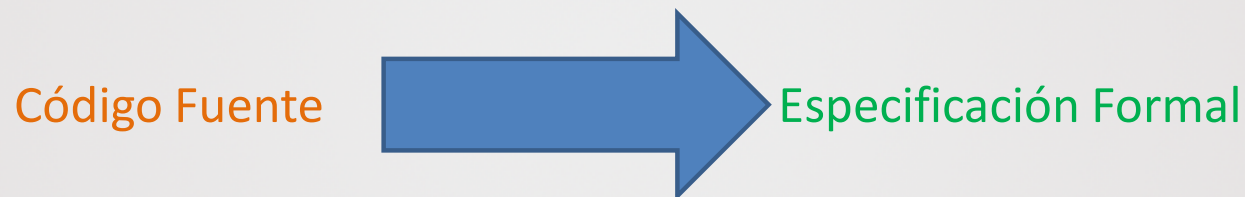
## Model checking



En la industria:

- Verificación de hardware  
Chequeo de limitaciones de la tecnología y su aplicación
- Verificación de software  
Software especializado: protocolos, sistemas de control  
Chequeo de abstracciones de código fuente

## Verificación Deductiva



Las pruebas establecen una relación de conformidad entre el código fuente y la especificación

En la industria:

- Verificación de hardware
  - Sistemas complejos, procesadores de punto flotante
- Verificación de software
  - Seguridad de sistemas críticos
  - Librerías
  - Implementación de protocolos