Hacia una Formalización de Sistemas Tolerantes a Fallas basada en LTS-Reactivos

Araceli Acosta

director

Nazareno Aguirre

Diciembre 2010

Motivación

- Diseño de *Sistemas críticos*:
  - *aeroespaciales*,
  - médicos,
  - de control de plantas nucleares,
  - de control de vehículos, etc.
- Existen eventos o *fallas* fuera del control de software
  - Rotura de sensores
  - Pérdida de mensajes
  - Alteraciones en los datos almacenados en una memoria
- La necesidad de tolerar estos eventos ha llevado al estudio de la *tolerancia a fallas*.

Metodología

- Los *métodos formales* han aportado numerosas herramientas para la verificación de software.
  - La *Confiabilidad* es uno de los atributos de calidad más importantes asociados al desarrollo de software; y sin duda el más importante para los sistemas críticos.
  - La *corrección de software* es, tal vez, el factor más importante asociado a la confiabilidad.
- Tratar la tolerancia a fallas desde etapas tempranas del desarrollo de software: *especificación y diseño*,
  - en contraposición con resetas *ad-hoc* que se utilizan comunmente en etapas avanzadas del desarrollo.
- Trabajaremos con la *hipótesis* de que *la fuente de fallas está fuera del control del sistema*

Objetivos generales

- Buscar *lenguajes* apropiados para la especificación y deseño de sistemas tolerantes a fallas que permitan:
  - *Facilitar la descripción* de la *ocurrencia* de fallas, sus *efectos* y *consecuencias* en el sistema
  - *Modularizar* la descripción y/o el diseño de los sistemas tolerantes a fallas
- Diseñar *métodos y herramientas* para el diseño y desarrollo de software con la característica de recuperación de fallas.

Terminología

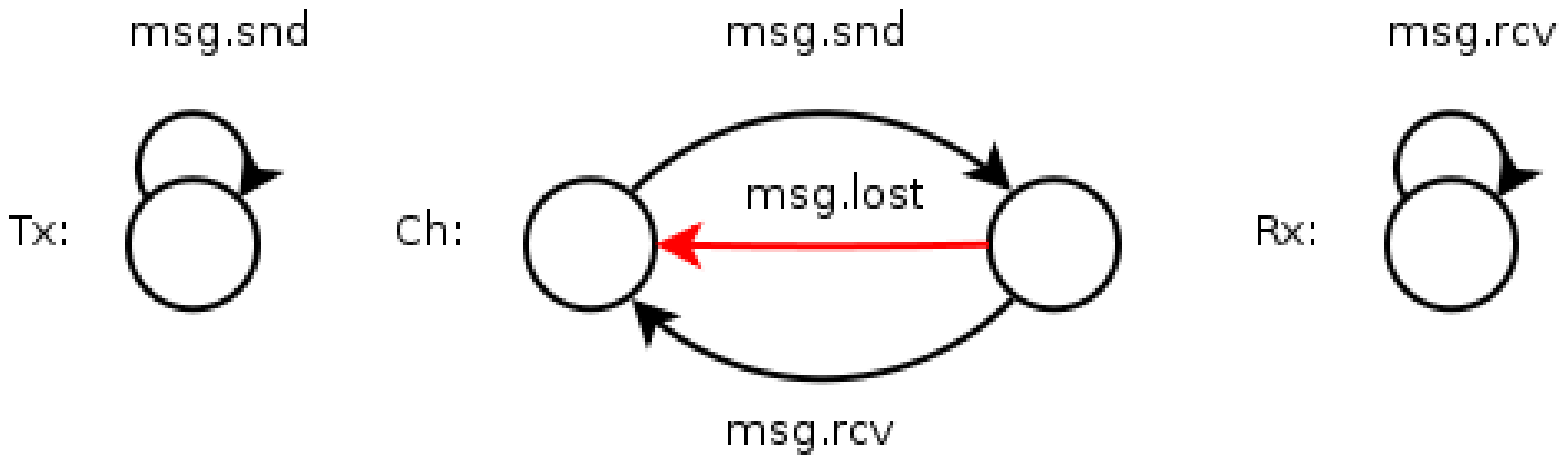
- un fallo (*failure*) es un comportamiento indeseado del sistema (es decir, una violación a alguno de los requisitos del sistema),
- un *error* es un estado del sistema que puede llevar a un fallo,
- una falla (*fault*) es un evento que puede llevar al sistema a un error.

Se denomina a un sistema tolerante a fallas si, incluso en presencia de fallas, el sistema se comporta de la manera esperada (es decir, sin violar requisitos de sistema).

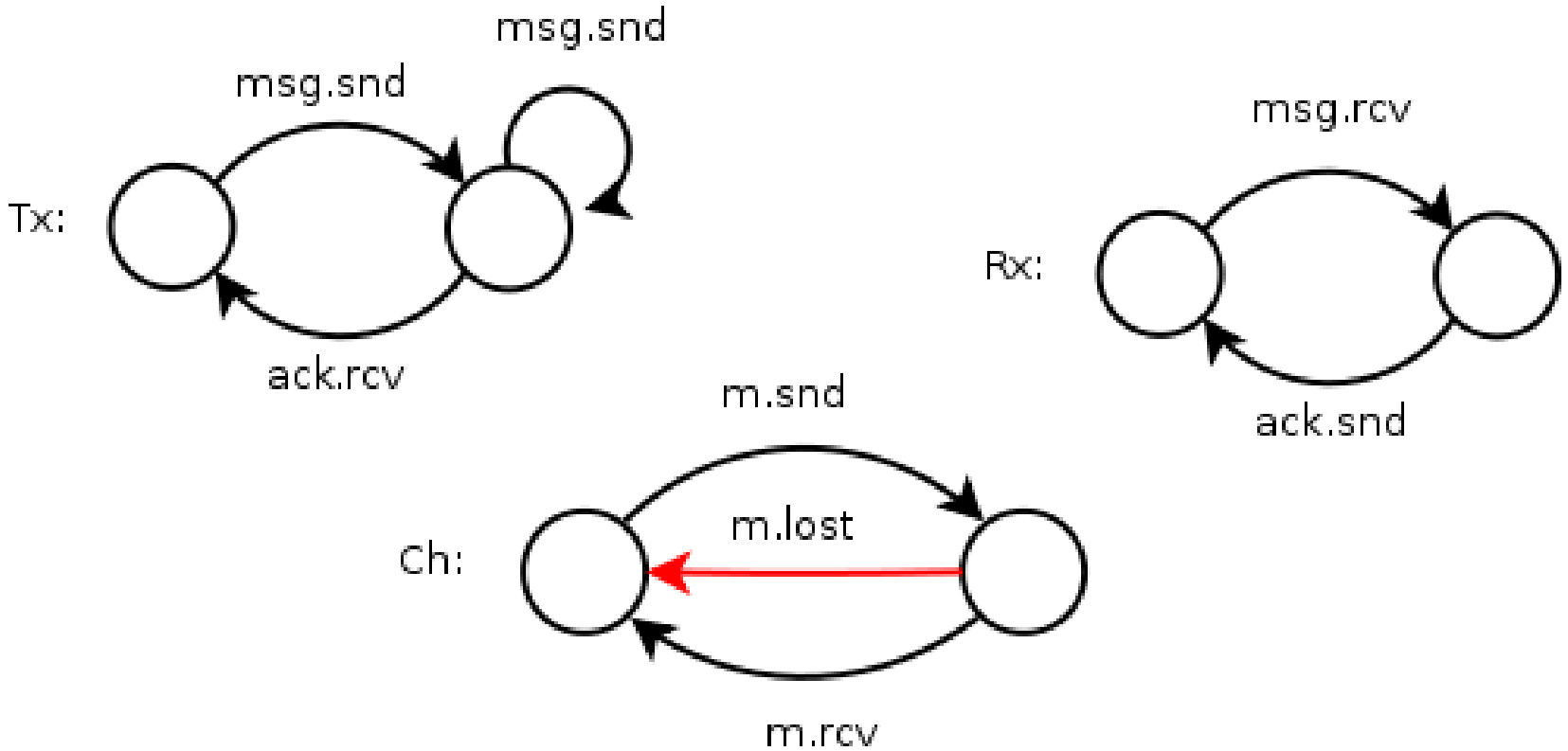
¿Por qué distinguimos las fallas de otros eventos?

Modularizar la descripción y/o el diseño de los sistemas tolerantes a fallas

- Diferenciar el comportamiento *normal* del sistema del comportamiento *anormal*.

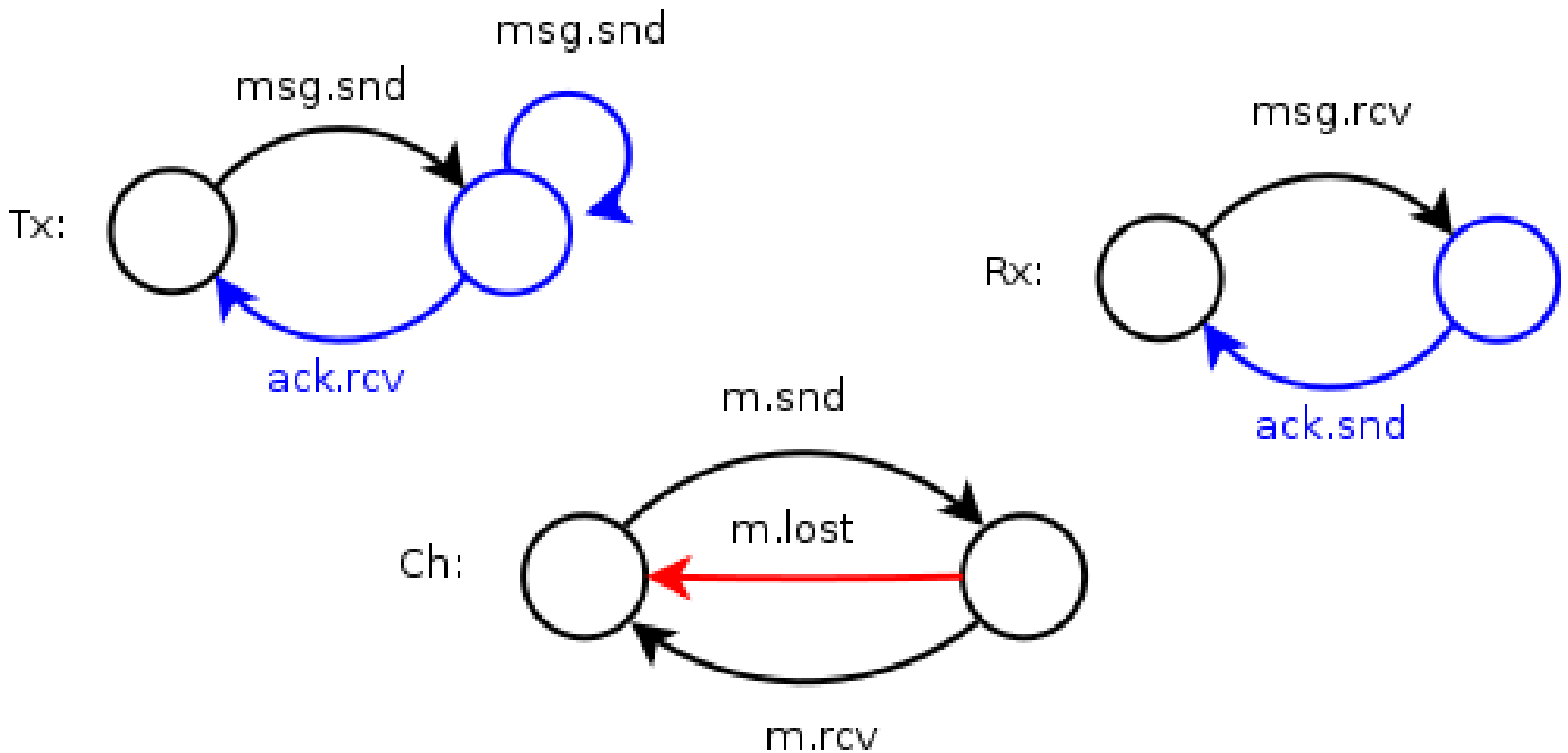


Sistema tolerantes a fallas

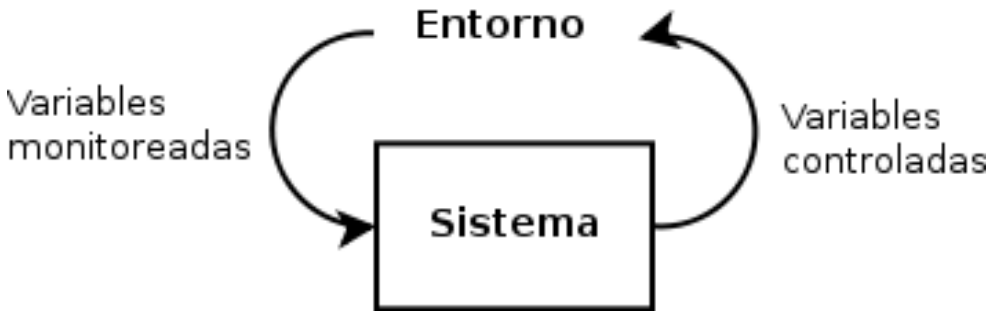


Desafío

A partir del diseño del *sistema sin fallas*, y la identificación de las *fallas* construir automáticamente el comportamiento faltante, es decir el sistema de recuperación.



Síntesis de controladores



$$Sistema? \;|||\; Entorno \models \varphi$$

- *Entorno* : modelo del comportamiento del ambiente
- φ : comportamiento del sistema-entorno que se quiere garantizar. *Especificación del sistema*.
- *Sistema*? : modelo del sistema. Esta es la *incognita*.

Problemas

- La *complejidad* para el caso general es 2EXP
  - Para fórmulas de la forma $G\ F\ assume \rightarrow G\ F\ goal$ el algoritmo es cúbico.
- El modelo del sistema debe ser *determinista*
- Las características particulares de las fallas respecto de otros eventos: *noción de fairness de fallas*
- Los límites en el modelo posible de la solución. Se cuenta con acciones *controladas* y acciones *observadas*, pero no se cuenta con acciones *no observables*.
- Poder especificar *requisitos blandos*, para refinar la búsqueda de soluciones más acorde los mismos.

Trabajo actual

- Plantear el problema de cálculo del *sistema de recuperación* en términos de la síntesis de controladores.
- Ver si es posible resolver de manera razonable la cuestión de las acciones *no observables* sin reducirlo a un problema no determinista.
- En su defecto trabajar con *requisitos blandos*.

FIN