Autómatas de protocolos

Redes de Comunicaciones - Curso 2024/25

Introducción

A la hora de diseñar un protocolo de comunicación, además del formato exacto de los mensajes, también necesitamos especificar cuál es la secuencia correcta de intercambio de los mismos, puesto que no todas las combinaciones de envío de mensajes serán posibles ni tendrán sentido. Durante esta sesión vamos a analizar la temporización o secuencia correcta del intercambio de mensajes de un protocolo, que normalmente suele especificarse mediante modelos computacionales como los autómatas de estados finitos.

Autómatas finitos

La forma habitual de concretar las secuencias válidas de intercambios de mensajes en un protocolo de comunicación es mediante modelos computacionales como los autómatas (o máquinas de estados) finitos. Dado que dichos modelos son vistos en detalle en otras materias de la titulación, así como en las clases teóricas de la asignatura de Redes de Comunicaciones, en esta sesión nos limitaremos a elaborar autómatas concretos basados en la práctica a desarrollar en la asignatura.

A modo de recordatorio, indicaremos que un autómata finito puede considerarse como una quíntupla (Q, Σ, δ, q_s, F) en la que:

- Q es el conjunto de estados.
- Σ es el alfabeto de entrada o secuencia de eventos.
- δ es el conjunto de transiciones entre los estados.
- q_s es el estado inicial del autómata.
- F es el conjunto de estados finales del autómata.

Se estudiarán diferentes autómatas que irán aumentando de complejidad conforme se introduzcan nuevos elementos y condiciones.

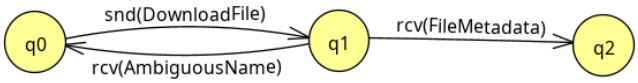
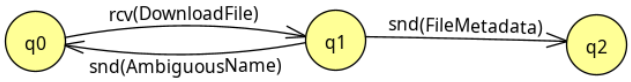
En esta asignatura, la especificación de los autómatas se realizará mediante el grafo de transiciones, que permite visualizar la quintupla por completo. Además, Σ se definirá en detalle cuando se definan los mensajes y su formato en la sección correspondiente de la documentación que los alumnos deben entregar con el protocolo diseñado.

Existen varias formas de plasmar un protocolo en forma de un autómata finito. Nosotros tomaremos una estrategia para el diseño del protocolo basada en:

- *eventos de interés* (p. ej., se envía un mensaje). El evento de envío de un cierto mensaje se denotará en el autómata mediante **snd(message_name)**, mientras que la recepción se denota con **rcv(message_name)**. Entre paréntesis se indicará el tipo de mensaje enviado o recibido (operación u *opcode*).
- *estados que requieren un procesamiento o una acción concreta* (p. ej., se agota el tiempo establecido en un temporizador sin haber recibido respuesta).

Autómatas para un protocolo cliente-servidor basado en canal confiable

En primer lugar, vamos a considerar el caso de un protocolo dado cuando el canal subyacente es confiable (e.g., TCP). Siguiendo la notación que se emplea en las clases teóricas para el análisis de los protocolos confiables, debemos considerar una representación del protocolo basada en dos autómatas; un autómata para el cliente y otro para el servidor. Ciertamente, ambos autómatas tienen estados comunes en la mayoría de los casos dado que deben mantener la secuencia de mensajes intercambiados. Por ejemplo, vamos a comenzar por dos autómatas que reflejan el comportamiento de un cliente cuando está solicitando descargar un fichero, tras haber ejecutado el comando *download* y haber establecido la conexión con uno de los servidores del fichero solicitado. Dichos autómatas serían:

Autómata para el rol cliente (receptor de ficheros)	**Autómata para el rol servidor de ficheros**
	

Como podemos comprobar, la única diferencia entre ellos, en este caso, está en quién es el encargado de enviar o recibir los mensajes. Cabe destacar que es posible recibir respuestas distintas a un cierto mensaje enviado (p.ej., recibir un mensaje *AmbiguousName* o bien recibir *FileMetadata* en respuesta al envío de un mensaje *DownloadFile*). Estas respuestas, además, podrían dejar al programa en un estado distinto. En estos ejemplos no se ha establecido estado inicial ni final(es) con la intención de que estas transiciones emularan una pequeña parte de un autómata más grande. Siendo estrictos, no son autómatas bien formalizados. Por otra parte, hay que tener en cuenta que no siempre el autómata del servidor se comportará como un reflejo del autómata cliente. Algunas peticiones pueden requerir que el servidor realice ciertas acciones que involucren eventos en la red que deben aparecer en su propio autómata con la secuenciación correcta.

Autómatas para un protocolo cliente-servidor con posibilidad de error y confirmaciones

En algunos casos, cuando el canal subyacente no es confiable, tenemos que tener en cuenta la posibilidad de que se pierdan algunos mensajes. Vamos a ver cómo se llevaría a cabo la especificación de los autómatas para esos casos.

Como es posible imaginar, la complejidad de los autómatas aumenta cuantas más condiciones de error debemos considerar. En este apartado vamos a ver el diseño de un autómata en el que se contempla que tanto las pérdidas como los errores tienen la misma consecuencia: el canal subyacente descarta el mensaje.

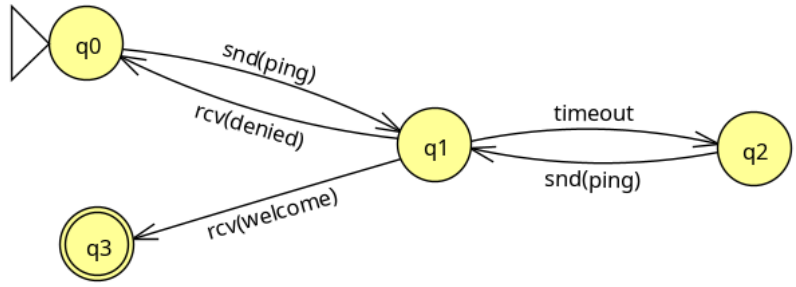
Vamos a utilizar como base el intercambio de mensajes entre el cliente de *NanoFiles* y el servidor de directorio que ya se empezó a trabajar en boletines anteriores.

Siguiendo la metodología de clase de teoría, un protocolo de parada y espera de este tipo que permita errores debe implementar dos mecanismos fundamentales:

- Envío de confirmaciones implícitas o explícitas.
- Uso de temporizadores (*timeouts*) de envío.

Este podría ser el autómata de un *peer* cliente para el envío del mensaje de *ping* al servidor de directorio:

Autómata del cliente de directorio



Como se puede apreciar, este autómata tiene varias características:

- El estado inicial y final son distintos. Esto impediría que, tras un *ping* exitoso, se permita realizar de nuevo un *ping* al directorio. Este tipo de decisiones de diseño quedan a discreción de los alumnos. En cualquier caso, es importante recalcar que en el diseño de los autómatas de las prácticas se debe buscar que cumplan con las siguientes restricciones:
  - Son autómatas mínimos.
  - Permiten seguir las iteraciones que admite el propio código que modela.
  - No aceptan ejecuciones que el código entregado no puede ejecutar.
- El mecanismo de *timeout* establecido es infinito. Esto es altamente indeseable. Se puede asumir que tras una serie de reintentos acotados en número, el servidor está caído y no va a responder. Al ser los autómatas unas máquinas de estados sin memoria, se recomienda a los alumnos definir un **evento de interés** llamado *timeout_n*, que se produciría con el evento de timeout n-ésimo con prioridad sobre el de *timeout* estándar, y que permitiría transitar a otro estado distinto.

Ejercicios a realizar

1. Basándote en el autómata del cliente de directorio mostrado anteriormente (disponible en el Aula Virtual como `ejemplo_no_confiable_NF_ping.jff`), amplíalo para dar cabida a la consulta de la lista de ficheros publicados por otros *peers* en el directorio, la publicación de ficheros los ficheros que un *peer* quiere compartir con el resto, o la consulta de los servidores que poseen un determinado fichero a descargar.

   - No olvides añadir las transiciones necesarias para incluir un mecanismo de *timeouts* y retransmisión, que haga que el protocolo con el directorio sea confiable a pesar de operar sobre un canal no confiable como UDP.
   - Recuerda que las decisiones de diseño acerca de las secuencias válidas de intercambio de mensajes quedan a discreción del alumnado. Un ejemplo de tal decisión sería si se hace obligatorio publicar los ficheros que un *peer* sirve al resto antes de poder obtener la lista de ficheros servidos por otros. Las decisiones de diseño del protocolo adoptadas por cada grupo deberán reflejarse en el autómata.

2. Diseña el resto del autómata del **cliente de directorio**, teniendo en cuenta las mejoras planteadas por cada grupo de prácticas para su posterior implementación. *NOTA: Considerar una mejora en esta fase del diseño no obliga a su implementación posterior, pero sí es requisito necesario*.

3. Diseñar el resto del autómata del **servidor de directorio**. Se puede considerar que el directorio siempre responderá a cualquier petición que se le haga, es decir, el directorio no mantiene un registro de las interacciones anteriormente realizadas con un determinado cliente (el directorio es *stateless*).

4. Diseñar los dos autómatas, cliente y servidor, para el protocolo de transferencia de fragmentos de ficheros entre *peers*. Dichos autómatas regirán la comunicación entre un *peer* servidor de ficheros que ha ejecutado el comando `serve`, y otro *peer* actuando como cliente, que ha ejecutado el comando `download`. Al diseñar este protocolo, se han de tener en cuenta aspectos como que un fichero solicitado puede no encontrarse entre los disponibles (p.ej., el nombre indicado no concuerda o bien es ambiguo), o que el cliente debe ser capaz de comprobar la integridad del fichero descargado (calculando su *hash* a partir del contenido y comparándolo con el *hash* indicado por el servidor). También hay que tener en cuenta en el diseño de este protocolo que la descarga se hará por fragmentos de un fichero (en vez del contenido íntegro del mismo), así como que se debe permitir descargar un fichero de múltiples servidores, con lo que será imprescindible asegurar que se trata del mismo fichero en todos los casos. Así, aunque en NanoFiles se utiliza el nombre del fichero como criterio para su localización y descarga, es importante tener en cuenta a la hora de diseñar el protocolo que los ficheros vienen identificados de manera unívoca por su *hash* (calculado a partir del contenido del fichero) y no por el nombre que puedan tener en cada servidor.

Bibliografía

- JFLAP Tutorial: http://www.jflap.org/tutorial/.
- Apuntes de la asignatura de Teoría de Autómatas.