Process and Data Modeling

Finite Automata, Communicating Finite Automata, and Model Checking

— A Formal Approach to Modeling

Jens Chr. Godskesen
IT University of Copenhagen
Overview

● Apetizer,
  – Why A formal approach?

● Finite Automata (FA), the basics

● Communicating FA’s
  – Concurrency
  – Non-determinismo

● Uppall
Why a formal approach to modelling? (I)

So far you have seen models that are informal, i.e. they have no formal semantics; and that’s fine for many purposes.

*Errors should be detected as early as possible*, i.e. during design. But, some errors are more subtle than others and hard for humans to detect. Well known errors (see aux. litt.) are:

- The crash of the Ariane 5 rocket (type conversion leading to an exception)
- The accidents of the Therac-25 radiation therapy machine (operative system race conditions)
- Mars Rover Pathfinder (mutual exclusion problem)

To let *software tools* help analysing models they may benefit if the models have a precise formal meaning.
Why a formal approach to modelling? (II)

Suppose two *concurrently* running processes P1 and P2, working with a shared object `shared` under **mutual exclusion**, i.e. `job1(shared)` and `job2(shared)` may not be carried out simultaneously.

Is Pettersons mutual exclusion algorithm defined below correct? We could ask a tool to check.

P1

```
while (true) {
    req1 = true;
    turn = 2;
    while (req2 && turn != 1);
    job1(shared);
    req1 = false;
}
```

P2

```
while (true) {
    req2 = true;
    turn = 1;
    while (req1 && turn != 2);
    job2(shared);
    req2 = false;
}
```
Why a formal approach to modelling? (III)

The tool **Uppall** has successfully been involved in verifying the following *industrial* cases:

- Mutual Exclusion Protocols
- Bang & Olufsen Audio/Video Protocol
- Philips Audio Protocol
- LEGO MINDSTORMS Systems

See the Uppall homepage ([www.uppaal.com](http://www.uppaal.com)) for more information.
A Riddle (I)

Consider a war scene at night with *four wounded soldiers*, an old wooden *bridge*, and a *torch*. The soldiers must cross the bridge to be safe.

- At most two soldiers may cross the bridge simultaneously.

- Because it’s night the torch is needed when crossing the bridge.

- The soldiers need 5, 10, 20, and 25 minutes respectively to cross the bridge.

How long time will it (at least) take the soldiers to be safe?
A Riddle (II)

We may try to solve the riddle ourselves, or we could take a model oriented view upon the problem and get help that way.

- *Specify* the problem (in our case as a set of communicating finite automata)

- *Analyse* (i.e. simulate or *verify*) the specification to solve the problem.

Let’s investigate how the **model checker** Uppall can help simulate and verify a model of the problem.
Finite Automata (I)

One of the most basic computations is to move from one state to another based on a given input stimuli, as carried out by many electromechanical devises, say:

- A Vending Machine
- Automatic Door Controller
- Elevator Controller
- Washing Machine
- Dishwasher
- …
Finite Automata (II)

Suppose an electronic door and two censors (rear and front). Initially the door is closed. If the door is closed and someone is in front of it and nobody behind it it must open. If it is open and nobody is in front or behind the door it must close.

A Finite Automaton (FA) for the door controller:
Finite Automata (III)

The FA can be represented by the following table (function)

<table>
<thead>
<tr>
<th></th>
<th>neither</th>
<th>front</th>
<th>rear</th>
<th>both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>Closes</td>
<td>Open</td>
<td>Closed</td>
<td>Closed</td>
</tr>
<tr>
<td>Open</td>
<td>Closed</td>
<td>Open</td>
<td>Open</td>
<td>Open</td>
</tr>
</tbody>
</table>
Finite Automata (IV)

Sometimes it’s convenient to let some states be accepting states.

Example: An identifier is a sequence of letters and digits starting with a letter, $ $ and _ (underscore) are letters. Let the state named Ok below be the accepting state.
Finite Automata (V)

A Finite Automaton (or a Finite-state Machine) $M$ is a five tupple

$$M = (Q, \Sigma, \rightarrow, q_0, F)$$

where

- $Q$ is a set of states,
- $\Sigma$ is a finite set of input symbols,
- $\rightarrow \subseteq Q \times \Sigma \times Q$ is the transition relation,
- $q_0$ is the initial state, and
- $F \subseteq Q$ is the set of accepting states.

If $(q, a, q') \in \rightarrow$ we say that there is an $a$-transition (or just a transition) from $q$ to $q'$ and write $q \xrightarrow{a} q'$. 
Finite Automata (VI)

Let $M = (Q, \Sigma, \rightarrow, q_0, F)$ be a FA and let $\omega = a_1a_2\ldots a_k$ be a string over $\Sigma$ (i.e. $\omega \in \Sigma^*$). $M$ accepts $\omega$ if there exists a path

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \ldots \xrightarrow{a_k} q_k$$

with $q_k \in F$.

The set of all strings $L$ that $M$ accepts is the language accepted by $M$.

**example:** The FA on slide 11 accepts all Java identifiers.

The languages accepted by FA's are called regular languages.
Finite Automata (VI)

Not all languages are regular! For instance,

\[ L = \{ \omega \in \{0, 1\}^* \mid \omega \text{ has equal number of 0's and 1's} \} \]

is not regular. Why?

To accept \( L \) a **Pushdown Automata** (PDA) is needed, i.e. a FA with a stack as memory. Some languages cannot be accepted by a PDA.

A **Turing Machine** (TM) is a FA equipped with an unlimited tape as memory. A TM is conjectured to be the formal equivalent of the intuitive notion of an *algorithm*; and no one has falsified that conjecture.
Finite Automata (VII)

Exercises

1. Define a FA accepting $L_1 = \{\omega \in \{0,1\}^*| \omega \text{ contains at least four 1's}\}$.

2. Define a FA accepting $L_2 = \{\omega \in \{0,1\}^*| \omega \text{ contains the substring 1010}\}$.

3. Define a FA representing the lifecycle of a book in a library. How may a borrower be defined?

4. Define a language, not being $\{\omega \in \{0,1\}^*| \omega \text{ has equal number of 0's and 1's}\}$, that is also non-regular.

5. Is $\{\omega \in \{0,1\}^*| \omega \text{ has an equal number of 01 and 10 as substrings}\}$ regular? Justify your answer.
Communicating Finite Automata (I)

As we saw in the introductory example of this lecture (the riddle) models may consist of communicating FA's.

A **communicating FA** (CFA) is a FA where each transition is either an **output**, an **input**, or an **internal** transition.

We write $q \xrightarrow{a!} q'$ for an output transition, $q \xrightarrow{a?} q'$ for an input transition, and $q \rightarrow q'$ for an internal transition.

A CFA is defined by (we omit the set of accepting states from now on): $M = (Q, \Sigma, \rightarrow, q_0)$ where $Q$, $\Sigma$, and $q_0$ are as before and

$$\rightarrow \subseteq (Q \times \Sigma \times \{!, ?\} \times Q) \cup (Q \times Q)$$
Communicating Finite Automata (II)

As an example, an (untimed) soldier may be defined by:

![Automaton Diagram]

- Unsafe
- Safe
- take!
- release!
- takeBack!
- release!
Communicating Finite Automata (III)

... and the torch may be defined by:

Let's run the system in Uppall.
Communicating Finite Automata (IV)

CFA’s run in parallel and hand-shake synchronize on (dual) input/output symbols (or labels).

Let \( M_1 = (Q_1, \Sigma, \rightarrow_1, q_1) \) and \( M_2 = (Q_2, \Sigma, \rightarrow_2, q_2) \) be CFA’s. The FA being the \textbf{composition} of \( M_1 \) and \( M_2 \) is

\[
M = (Q_1 \times Q_2, \Sigma, \rightarrow, (q_1, q_2))
\]

where \( \rightarrow \) is defined by

\[
\begin{align*}
p \overset{a_1}{\rightarrow_1} p' & \quad q \overset{a?}{\rightarrow_2} q' \\
(p, q) & \overset{a}{\rightarrow} (p', q') \\
q \overset{a_1}{\rightarrow_2} q' & \quad p \overset{a?}{\rightarrow_1} p' \\
(p, q) & \overset{a}{\rightarrow} (p', q')
\end{align*}
\]

\[
\begin{align*}
p \overset{}{\rightarrow_1} p' & \\
(p, q) & \overset{}{\rightarrow} (p', q)
\end{align*}
\]

\[
\begin{align*}
q \overset{}{\rightarrow_2} q' & \\
(p, q) & \overset{}{\rightarrow} (p, q')
\end{align*}
\]
Communicating Finite Automata (V)

The composition of CFA’s may be generalized to more than two CFA’s, i.e. **networks** of CFA’s.

Let $M_i = (Q_i, \Sigma, \rightarrow_i, q_i)$, $i = 1, \ldots, k$, be CFA’s. The composition is $M = (Q_1 \times \ldots \times Q_k, \Sigma, \rightarrow, (q_1, \ldots, q_k))$ where $\rightarrow$ is defined by

$$\frac{p_i \overset{a^1}{\rightarrow}_i p_i'}{a} \quad \frac{p_j \overset{a^2}{\rightarrow}_j p_j'}{a} \quad i < j$$

$$\frac{(p_1, \ldots, p_k) \overset{a}{\rightarrow} (p_1, \ldots, p_i', p_j', \ldots, p_k)}{i < j}$$

$$\frac{p_i \overset{a^2}{\rightarrow}_i p_i'}{a} \quad \frac{p_j \overset{a^1}{\rightarrow}_j p_j'}{a} \quad i < j$$

$$\frac{(p_1, \ldots, p_k) \overset{a}{\rightarrow} (p_1, \ldots, p_i', p_j', \ldots, p_k)}{i < j}$$

$$\frac{p_i \overset{a}{\rightarrow}_i p_i'}{a} \quad \frac{(p_1, \ldots, p_k) \overset{a}{\rightarrow} (p_1, \ldots, p_i', \ldots, p_k)}{i < j}$$
Communicating Finite Automata (VI)

Components of a *network* only communicate internally, i.e. in order for an output action to be executed a corresponding input action must be enabled and vice versa. A network is a **closed systems**.

This is in contrast to a FA that is expected to react on input stimuli from its environment. A FA and an model of its environment may be modelled explicitly as CFA’s in a closed system.

If a system in a given state has either more than one internal transitions or if several transitions with the same label are enabled we say that the system is **non-deterministic**.

The untimed version of our riddle is non-deterministic.
Uppall, modeling (I)

**Uppall** is a tool in which we can model CFA’s using a graphical editor. In Uppall terminology a CFA is often referred to as a **process**.

Essentially Uppall models consists of:

- global declarations of channels (labels),
- a parameterized template declaration for each type of CFA,
- a definition of each CFA based on its template, and
- a system definition.

There is substantial online help.

Let’s consider the model of the (untimed) riddle.
Uppall, Simulation (II)

Start the **simulator** by clicking on the simulator tab.

You may do either a random simulation, choose to step through a simulation yourself, or replay a previous simulation.

Each enabled transition is listed as either a pair of transition involved in a handshake communication, or as a single state in a process that is reachable by an internal transition.

The result of the simulation is displayed as a MSC and the current state of each CFA is displayed together with a proposed enabled transition (if any).
Uppall, Verification (III)

Simulation is a manual process where the path of transitions followed is decided by the user, or perhaps randomly by the simulation tool.

E.g., if we would like to validate as to whether it’s possible for the four soldiers in our riddle to be safe we must (more or less systematically) try to select a simulation path where eventually all soldiers have crossed the bridge.

In more realistic models it may be an almost impossible task to make sure by simulation that a certain state in a composed system is reachable.

With the help of a verification tool we can have the states of a model explored automatically.
Uppall, Verification (IV)

The verifier is started clicking the verifier tab. Verification is carried out issuing \textit{queries} to the system.

For instance, in our riddle, we would like to reach a state along a path where the following \textit{proposition} holds:

\texttt{Soldier1.Safe and Soldier2.Safe and Soldier3.Safe and Soldier4.Safe}

A query stating that some proposition \( p \) holds in some state is written \( E<> p \). Let's ask Uppall the query:

\( E<> \texttt{Soldier1.Safe and Soldier2.Safe and Soldier3.Safe and Soldier4.Safe} \)

If we choose as an option, to have a \textit{diagnostic trace} generated we can replay the solution in the simulator.
Uppall, Verification (V)

Instead of asking about a proposition being valid in some state we may be interested in a proposition being true in all states the system can reach.

As an example in our riddle, it should always be the case that the torch is on the safe side only if at least one of the soldiers is also there:

A[] Torch.Safe imply
   (Soldier1.Safe or Soldier2.Safe or Soldier3.Safe or Soldier4.Safe)

but it’s not the case that just because one soldier is on the safe side then one of the other soldiers is also safe:

A[] Soldier1.Safe imply (Soldier2.Safe or Soldier3.Safe or Soldier4.Safe)
Uppall, Verification (IV)

The queries are interpreted relative to the *traces* the FA being the composition of the CFA's in the model can execute. A trace is some finite sequence

\[ q_0 \rightarrow^* a_1 \rightarrow^* q_1 \rightarrow^* a_2 \rightarrow^* \ldots \rightarrow^* a_k \rightarrow^* q_k \]

\( \exists \) means that there exists a *trace*, and \( \forall \) means *for all traces*. \( < > \) means *for some state* in a trace, and \( [ ] \) means *for all states* in a trace.

The standard query operators Uppall allows are:

- \( \exists <> \) P: P holds in some reachable state.
- \( \forall [ ] \) P: P holds in every reachable state.
- \( \exists [ ] \) P: P holds in every state along some path.
- \( \forall <> \) P: P holds in some state along every path.
Uppall, Exercises

Exercises

1. Don’t forget the FA exercises on slide 15

2. If Uppall is not installed on your computer go to the homepage of Uppall, www.uppall.com, follow the instructions for downloading Uppall and install it.

3. Define a book CFA and a borrower CFA and let the two constitute a closed system in Uppall. Try to understand your model by simulation. Then, extend your model to contain two borrowers and check that they cannot both have borrowed the book at the same time.

4. Define two CFA’s, one being a vending machine selling coffee and one being a customer buying coffee. Let the coffee machine require 7 Dkr. pr. cup of coffee. The input to the coffee machine should be coins of Dkr. 1, 2, and 5. By convenience, let the user always be willing to insert any coin and receive coffee. Try to understand your model by simulation.
5. Define relevant queries to your system from exercise 4 and check the queries in Uppall. For instance, it should be possible for the user to get a cup of coffee. Does Uppall find the the same trace depending on whether you choose to find a shortest diagnostic trace or just some trace?

6. Extend your system from exercise 3 to contain more than one book, and let a borrower be able to borrow more than one book. What about queries.

7. Let your system from exercise 4 consists of two users, make sure that buying coffee is carried out as a critical section. How to validate the last property? Does your queries from before still hold.