Embedded Systems Development

Lecture 2
Finite Automata & SyncCharts

Daniel Kästner
AbsInt Angewandte Informatik GmbH
kaestner@absint.com
Some things I forgot to mention

- Remember the HISPOS registration until 01.12. (Bachelor/Master CS+CuK).
- My email address: kaestner@cs.uni-sb.de
- Office in Science Park 1, AbsInt GmbH: 3rd floor, room 339 (with appointment).
Today

- Finite State Automata (FSA)
  - Definition
  - Deterministic vs. non-deterministic FSA
  - DFA minimization
  - Mealy/Moore automata

- SyncCharts
Model-based Software Development

- Lustre programs
- Esterel programs
- C Code
- Binary Code

Esterel SCADE
- SCADE language
- SyncCharts

aiT WCET Analyzer
- Timing Validation

SymTA/S
- System-level Schedulability Analysis
Model-based Software Development

SyncCharts as Enhancement to FSA

Esterel SCADE
- SCADE language
- SyncCharts
Model-based Software Development

Esterel SCADE
- SCADE language
- SyncCharts

Lustre programs
Esterel programs

C Code

Automata Minimization

Generator

Compiler

Compiler
Embedded Systems

- Typically, embedded systems are reactive systems:

  „A reactive system is one which is in continual interaction with its environment and executes at a pace determined by that environment“ [Bergé, 1995]

  Behavior depends on input and current state.

- automata model appropriate
Finite Automata
Finite Automata

- **Non-deterministic** finite automaton (NFA):
  \[ M = (\Sigma, Q, \Delta, q_0, F) \] where
  - \( \Sigma \): finite alphabet
  - \( Q \): finite set of states
  - \( q_0 \in Q \): initial state
  - \( F \subseteq Q \): final states
  - \( \Delta \subseteq Q \times (\Sigma \cup \{ \epsilon \}) \times Q \)

- \( M \) is called a **deterministic** finite automaton, if \( \Delta \) is a partial function

\[ \delta : Q \times \Sigma \rightarrow Q \]
Simple State Transition Diagram

- Used to represent a finite automaton
- Nodes: states
- $q_0$ has special entry mark
- Final states are doubly circled
- An edge from $p$ to $q$ is labelled by $a$ if $(p, a, q) \in \Delta$
- Example: integer and real constants:
Language Accepted by an Automaton

- $M = (\Sigma, Q, \Delta, q_0, F)$
- For $q \in Q, w \in \Sigma^*$: $(q, w)$ is a configuration.
- Binary step relation $\mathcal{M}$ on configurations:
  
  $$(q, aw) \mathcal{M} (p, w) \iff (q, a, p) \in \Delta : Q \times \Sigma \to Q$$
- Reflexive transitive closure of $\mathcal{M}$ is denoted by $\mathcal{M}^*$
- Language accepted by $M$:
  
  $$L(M) = \{ w \mid w \in \Sigma^*, \exists q_f \in F : (q_0, w) \mathcal{M}^* (q_f, \varepsilon) \}$$
Regular Languages / Expressions

- Let $\Sigma$ be an alphabet. The regular languages are defined inductively over $\Sigma$ by:
  - $\emptyset, \{\varepsilon\}$ are regular languages over $\Sigma$
  - For all $a \in \Sigma$, $\{a\}$ is a regular language
  - If $R_1$ and $R_2$ are regular languages over $\Sigma$, then also $R_1 \cup R_2, R_1R_2, R_1^*$.

- Regular expressions over $\Sigma$ are defined by:
  - $\emptyset$ is a regular expression and describes the language $\emptyset$
  - $\varepsilon$ is a regular expression and describes the language $\{\varepsilon\}$
  - $a$ (for $a \in \Sigma$) is a regular expression and denotes $\{a\}$
  - $(r_1|r_2)$ is a regular expression over $\Sigma$ and denotes $R_1 \cup R_2$
  - $(r_1r_2)$ is a regular expression over $\Sigma$ and denotes $R_1R_2$
  - $(r_1)^*$ is a regular expression over $\Sigma$ and denotes $R_1^*$.
Regular Expressions and FA

- For every regular language \( R \), there exists an NFA \( M \), such that \( L(M) = R \).

- Constructive Proof (Subset Construction):
  - A regular language is defined by a regular expression \( r \)
  - Construct an NFA with one final state, \( q_f \) and the transition

- Decompose \( r \) and develop the NFA according to the following rules -> until only transitions under single characters and \( \varepsilon \) remain.
Example: $a(a|0)^*$
Nondeterminism

- Sources of nondeterminism:
  - many transitions may be possible under the same character in a given state
  - $\varepsilon$-moves (next character is not read) may compete with non-$\varepsilon$-moves

- DFA:
  - No $\varepsilon$-transition
  - At most one transition from every state under a given character, ie for every $q \in Q$, $a \in \Sigma$:

\[ | \{ q' \mid (q, a, q') \in \Delta \} | \leq 1 \]
NFA -> DFA

- Let \( M = (\Sigma, Q, \Delta, q_0, F) \) be an NFA and let \( q \in Q \). The set of \( \epsilon \) successor states of \( q \), \( \epsilon\)-SS, is

\[
\epsilon\text{-SS}(q) = \{ p \mid (q, \epsilon) \in \Delta^*(p, \epsilon) \}
\]

or the set of all states \( p \), including \( q \), for which there exists an \( \epsilon \) path from \( q \) to \( p \) in the transition diagram for \( M \).

We extend \( \epsilon\)-SS to sets of states \( S \subseteq Q \):

\[
\epsilon\text{-SS}(S) = \bigcup_{q \in S} \epsilon\text{-SS}(q)
\]
NFA -> DFA

- If a language $L$ is accepted by a NFA then there is also a DFA accepting $L$.

- Let $M = (\Sigma, Q, \Delta, q_0, F)$ be an NFA. The DFA associated with $M$, $M' = (\Sigma, Q', \delta, q_0', F')$ is defined by:
  - $Q' \subseteq P(Q)$
  - $q_0' = \varepsilon\text{-SS}(q_0)$
  - $F' = \{S \subseteq Q \mid S \cap F \neq \emptyset\}$
  - $\delta(S, a) = \varepsilon\text{-SS}(\{p \mid (q, a, p) \in \Delta \text{ for } q \in S\})$ for $a \in \Sigma$

- Thus, the successor state $S$ under a character $a$ in $M'$ is obtained by combining the successor states of all states $q \in S$ under $a$ and adding the $\varepsilon$ successor states.
Algorithm NFA→DFA

$q'_0 := ε$-SS($q_0$); $Q' := \{q'_0\}$;
marked($q'_0$):=false; $δ := \emptyset$;

while $∃ S ∈ Q'$ and marked($S$)=false do
    marked($S$):=true;
    foreach $a ∈ Σ$ do
        $T := ε$-SS($\{p ∈ Q | (q, a, p) ∈ Δ$ and $q ∈ S\}$);
        if $T ∉ Q'$
            $Q' := Q' ∪ \{T\}; // new state$
            marked($T$):=false
        $δ := δ ∪ \{(S, a)→T\}; // new transition$
Example: $a(a|0)^*$
DFA Minimization

- After NFA->DFA the DFA need not have **minimal size**, i.e., minimal number of states and transitions.
- p and q are **undistinguishable**, iff for all words w both (q,w) and (p,w) lead by $\mathcal{L}_{M}^{*}$ into either $F'$ or $Q' - F'$.
- Undistinguishable states can be **merged**.
DFA Minimization

- **Input:** DFA \( M = (\Sigma, Q, \delta, q_0, F) \)
- **Output:** DFA \( M_{\text{min}} = (\Sigma, Q_{\text{min}}, \delta_{\text{min}}, q_{0\text{min}}, F_{\text{min}}) \) with \( L(M) = L(M_{\text{min}}) \) and \( Q_{\text{min}} \) minimal.
- Iteratively refine a partition of the set of states where each set \( S \) in the partition consists of states so far undistinguishable.
- Start with the partition \( \Pi = \{ F, Q - F \} \).
- Refine the current \( \Pi \) by splitting sets \( S \in \Pi \) into \( S_1, S_2 \) if there exist \( q_1, q_2 \in S \) such that
  - \( \delta(q_1, a) \in S_1 \)
  - \( \delta(q_2, a) \in S_2 \)
  - \( S_1 \neq S_2 \)
- Merge sets of undistinguishable states into a single state.
Algorithm minDFA

\[ \Pi := \{F, Q-F\} \]
do changed := false
\[ \Pi' := \Pi; \]
foreach K in \( \Pi \) do
\[ \Pi' := (\Pi' - \{K\}) \cup \{\{K_i\}_{1 \leq i \leq n}\} \text{ with } K_i \text{ maximal such that } \]
\[ K = \bigcup_{1 \leq i \leq n} K_i \text{ and } \forall a \in \Sigma \forall q \in K_i \exists K'_i \in \Pi : \delta(q, a) \in K'_i \]
if \( n > 1 \) then changed := true fi
\[ \Pi := \Pi'; \]
until not changed;
\[ Q_{\text{min}} = \Pi - (\text{Dead } \cup \text{ Unreachable}); \]

\[ q_{0\text{min}} \quad \text{Class of } \Pi \text{ containing } q_0 \]
\[ F_{\text{min}} \quad \text{Classes containing an element of } F \]
\[ \delta_{\text{min}}(K,a)=K' \text{ if } \delta(q,a)=p \text{ with } a \in \Sigma \text{ and } p \in K' \text{ for one (ie for all) } q \in K \]
\[ K \in \text{Dead} \quad \text{if } K \text{ is not final state and contains only transitions to itself} \]
\[ K \text{ Unreachable} \quad \text{if there is no path from the initial state to } K \]
Example: $a(a|0)^*$
Mealy Automata

- Mealy automata are finite-state machines that act as transducers, or translators, taking a string on an input alphabet and producing a string of equal length on an output alphabet.
- A machine in state $q_j$, after reading symbol $\sigma_k$, writes symbol $\lambda_k$; the output symbol depends on the state just reached and the corresponding input symbol.
- A Mealy automaton is a six-tuple $M_E=(Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ where
  - $Q$ is a finite set of states
  - $\Sigma$ is a finite input alphabet
  - $\Gamma$ is a finite output alphabet
  - $\delta: Q \times \Sigma \rightarrow Q$ is the transition function
  - $\lambda: Q \times \Sigma \rightarrow \Gamma$ is the output function
  - $q_0$ is the initial state
Moore Automata

- Moore automata are finite-state machines that act as transducers, or translators, taking a string on an input alphabet and producing a string of equal length on an output alphabet.
- Symbols are output after the transition to a new state is completed; output symbol depends only on the state just reached.

A Moore automaton is a six-tuple $M_O=(Q, \Sigma, \Gamma, \delta, \lambda, q_0)$ where

- $Q$ is a finite set of states
- $\Sigma$ is a finite input alphabet
- $\Gamma$ is a finite output alphabet
- $\delta: Q \times \Sigma \rightarrow Q$ is the transition function
- $\lambda: Q \rightarrow \Gamma$ is the output function
- $q_0$ is the initial state
State Transition Diagrams

- **State transition**: When event $\gamma$ occurs in state A, if Condition P is true at the time, the system executes action $a$ and transfers to state C.
- State diagrams are directed graphs with nodes denoting states, and arrows (labelled with the triggering event, guarding conditions and action to be executed) denoting transitions.
- Example:

![State Transition Diagram](image)

- Problem: all combinations of states have to be represented explicitly, leading to exponential blow-up.
State Transition Diagrams

- **Disadvantages:**
  - No **structure** (no strategy for bottom-up of top-down development)
  - State-transition diagrams are **flat**, i.e., without hierarchy
  - **Uneconomical wrt transitions** (e.g., interrupt): exponential blow-up
  - **Uneconomical wrt states:** exponential blow-up
  - **Uneconomical wrt parallel composition:** exponential blow-up
  - **Inherently sequential:** parallelism cannot be expressed in a natural way
SyncCharts

- Visual formalism for describing states and transitions of a system in a modular fashion.

- Extension of state-transition diagrams (Mealy/Moore automata):
  - Hierarchy
  - Modularity
  - Parallelism

- Is fully deterministic.

- Tailored to control-oriented applications (drivers, protocols).

- Implements synchronous principle.
Synchronous Programming

- Two simple ways of implementing reactive systems:
  - Event-driven
    ```cpp
    <Initialize Memory>
    Foreach input_event do
      <Compute Outputs>
      <Update Memory>
    End
    ```
  - Sampling
    ```cpp
    <Initialize Memory>
    Foreach period do
      <Read Inputs>
      <Compute Outputs>
      <Update Memory>
    End
    ```
Synchronoous Programming

- Program typically implements an automaton:
  - **state**: valuations of memory
  - **transition**: reaction, possibly involving many computations

- **Synchronous paradigm**: reactions are considered atomic, ie they take no time. (Computational steps execute like combinatorial circuits.)

- **Synchronous broadcast**: instantaneous communication, ie each automaton in the system considers the outputs of others as being part of its own inputs.
Synchronous Programming

- Important requirement: guaranteeing deterministic behavior.

- Time is divided into discrete ticks (also called cycles, steps, instants).

- Implicit assumption: presence of a global clock. This makes application in distributed environments difficult.

- In order to validate the timing behavior it is sufficient to prove that the worst-case execution time (WCET) of any reaction is smaller than the minimal time interval between two external events.
Overview

- **StateCharts:**
  - First, and probably most popular formal language for the design of reactive systems.
  - Focus on specification and design, not designed as a programming language.
  - Determinism is not ensured.
  - No standardized semantics.

- Programming languages for designing reactive systems:
  - **ESTEREL** [Berry]: textual imperative language.
  - **SyncCharts / SSM**: Graphical formalism for ESTEREL.
  - **LUSTRE** [Caspi, Halbwachs]: textual declarative language. Tailored to data-flow oriented systems (e.g. regulation systems).
  - **SCADE** [Esterel Inc.]. Graphical formalism for (enhanced) LUSTRE.
Cyclic Evolution

- Reactions consist of three phases:
  1. Read input signals (input event)
  2. Compute the reaction
  3. Perform outputs (output event)

- All phases happen in one instant, ie have 0-duration.
SyncCharts

- **States (circles and rectangles):**
  - can be named
  - two types:
    - *simple state* (circle)
    - *macrostate* (rounded rectangle): contain a hierarchy of other states
  - are optionally labelled*: /<effect>

- **Transitions (arrows):**
  - are labelled*: <trigger>/<effect>
    - All components are optional.
  - three types:
    - strong abort
    - weak abort
    - normal termination
  - can have priorities (→ determinism)

*Triggers and effects are signals, or combinations of signals using boolean operations *or*, *and* and *not.*
States & State Transition Graphs

- Special states:
  - Initial state: $s$ (alternative notation: $\text{Initial state: } s$)
  - Terminal state: $\bigcirc$

- State Transition Graph: connected labeled graph made of states connected by transitions, with an initial state.

- Two types of states:
  - Simple state: just carries a label.
  - Macrostate: contains at least one state transition graph.

- At each instant there is one and only one active state.
- An active state waits for the satisfaction of the trigger of one of its outgoing transitions, at an instant *strictly posterior* to its entering (activation).
State and Transition Labels

- Signals are characterized by their presence status (+, -, ⊥).
  - Valued signal: signals conveys a value of a given type.
  - Pure signal: no value conveyed.
- tick: implicit signal present at every instant.
- A trigger is satisfied ⇔ associated signal is present.
- Transition labels:
  - When the trigger is satisfied, the transition is said to be enabled.
  - The transition is immediately taken and emits the associated signals.
  - The firing of a transition is fully deterministic and takes no time.
- Node labels:
  - Signal emission depends on transition type (strong/weak abort)
  - Signals are emitted when...

<table>
<thead>
<tr>
<th>Weak abort</th>
<th>...entering</th>
<th>...in</th>
<th>...exiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Strong abort</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Example: Strong vs. Weak Abort

<table>
<thead>
<tr>
<th>Instant</th>
<th>Input</th>
<th>TFF-SA Output</th>
<th>TFF-WA Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: Strong vs. Weak Abort

<table>
<thead>
<tr>
<th>Instant</th>
<th>Input</th>
<th>TFF-SA Output</th>
<th>TFF-WA Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>C, OFF</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>OFF</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td>C, OFF</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>ON</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>ON</td>
<td></td>
</tr>
</tbody>
</table>
Example: Strong vs. Weak Abort

<table>
<thead>
<tr>
<th>Instant</th>
<th>Input</th>
<th>TFF-SA Output</th>
<th>TFF-WA Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>ON</td>
<td>OFF,ON</td>
</tr>
<tr>
<td>3</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>C, OFF</td>
<td>ON,C,OFF</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>OFF</td>
<td>OFF</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>ON</td>
<td>OFF,ON</td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td>C,OFF</td>
<td>ON,C,OFF</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>ON</td>
<td>OFF,ON</td>
</tr>
<tr>
<td>9</td>
<td>ON</td>
<td>ON</td>
<td>ON</td>
</tr>
</tbody>
</table>
Concurrent

- A macrostate can contain a parallel composition of separate concurrent STGs. Graphical notation: dashed separation line.
- STGs are coupled by shared signals.
- A local signal is declared by the keyword signal and its scope is the containing macrostate.
- A set of (concurrent) active states is called a configuration.

Notation:
- Active state \( S^+ \): presence of signal \( S \) \( S^- \): absence of signal \( S \)
Example Reaction
Concurrency and Normal Termination

- When each concurrent STG in a macrostate reaches a final state, then the macrostate is immediately exited by its normal termination transition.
Concurrency and Abort

ABRO

ABO

WaitAandB

wA

wB

A

B

dA

dB

/O

done

R

R+ B+

ABRO

ABO

WaitAandB

wA

wB

A

B

dA

dB

/O

done

R
Transitions

- A strong abort prevents any execution in the preempted state.

- For any state
  - every outgoing transition has a different priority
  - any strong abort transition has priority over any weak abort transition
  - any weak abort transition has priority over a normal termination transition

- There are no inter-level transitions.