Modellierung von Echtzeitsystemen

Reaktive Systeme

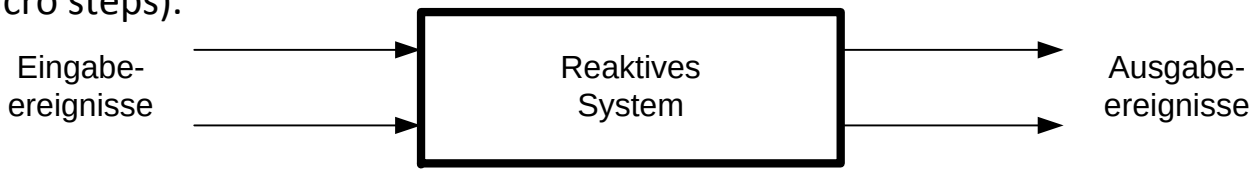
Werkzeuge: SCADE, Esterel Studio

Esterel

- Esterel ist im klassischen Sinne eher eine Programmiersprache, als eine Modellierungs-sprache
- Esterel wurde von Jean-Paul Marmorat und Jean-Paul Rigault entwickelt um die Anforderungen von Echtzeitsystemen gezielt zu unterstützen:
  - direkte Möglichkeit zum Umgang mit Zeit
  - Parallelismus direkt in der Programmiersprache
- G. Berry entwickelt die formale Semantik für Esterel
- Es existieren Codegeneratoren zur Generierung von u.a. **sequentiellen** C, C++ Code:
  - In Esterel werden (parallele) Programme in **einen** endlichen Automaten umgewandelt
  - Aus dem endlichen Automaten wird ein Programm mit **einem** Berechnungsstrang erzeugt ⇒ deterministische Ausführung trotz paralleler Modellierung.
- SCADE (ein kommerzielles Tool, dass u.a. die Esterel-Sprache verwendet) wurde bei der Entwicklung von Komponenten für den Airbus A380 eingesetzt.
- Ein frei verfügbarer Esterel-Compiler kann unter http://www-sop.inria.fr/esterel.org/files/ bezogen werden (siehe Links auf der Vorlesungs-Homepage).

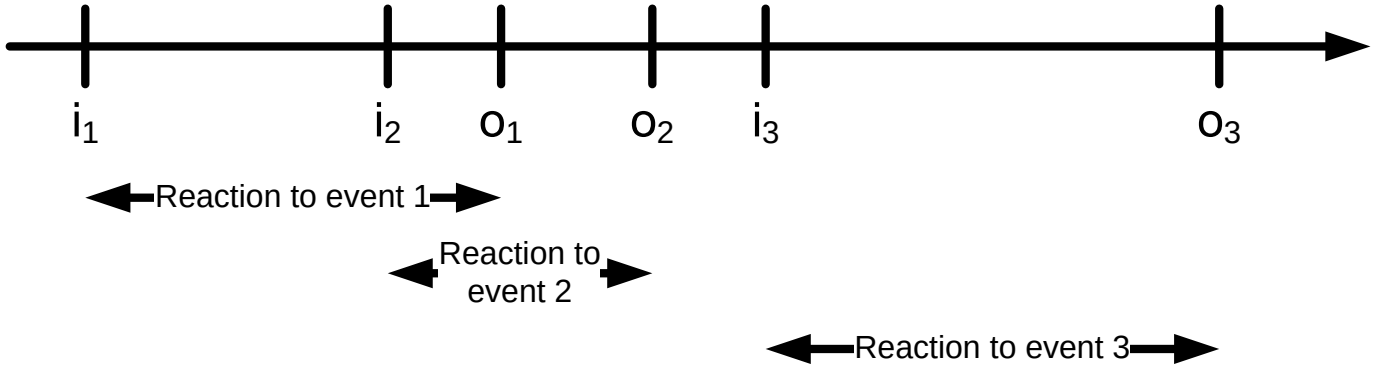
Einführung in Esterel

- Esterel gehört zu der Familie der **synchronen** Sprachen. Dies sind Programmiersprachen, die optimiert sind, um **reaktive Systeme** zu programmieren. Weitere Vertreter: Lustre, Signal, Statecharts
- Bei **reaktiven Systemen** erfolgen Reaktionen direkt auf **Eingabeereignisse**
- Synchrone Sprachen zeichnen sich vor allem dadurch aus, dass
  - Interaktionen (Reaktionen) des Systems mit der Umgebung die Basisschritte des Systems darstellen (**reaktives System**).
  - Anstelle von physikalischer Zeit die **logische Zeit** (die Anzahl der Interaktionen) verwendet wird.
  - Interaktionen, oft auch **macro steps** genannt, bestehen aus einzelnen Teilschritten (micro steps).



Reaktive Systeme

- In reaktiven Systemen (reactive / reflex systems) werden für Eingabeereignisse Ausgaben unter Einhaltung zeitlicher Rahmenbedingungen erzeugt.
- Reaktive Systeme finden u.a. Anwendung in der Industrie zur Prozesssteuerung und zur Steuerung / Regelung in Automobilen und Flugzeugen.
- Schwerpunkte bei der Umsetzung von reaktiven Systemen sind Sicherheit und Determinismus.
- Bearbeitung der Ereignisse kann sich überlappen (i input, o output)



Synchronitätshypothese

- Die Synchronitätshypothese (synchrony hypothesis) nimmt an, dass die zugrunde liegende physikalische Maschine des Systems unendlich schnell ist.

  → Die Reaktion des Systems auf ein Eingabeereignis erfolgt augenblicklich (ohne erkennbare Zeitverzögerung). Reaktionsintervalle reduzieren sich zu Reaktionsmomenten (reaction instants).

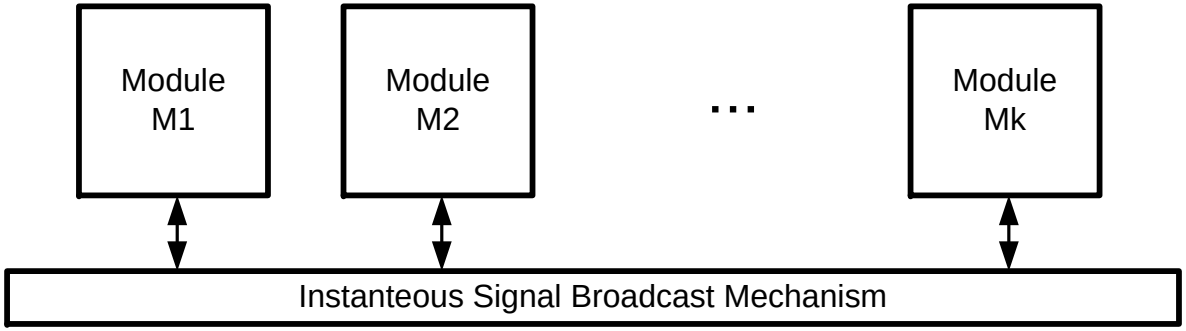
- **Rechtfertigung**: Diese Annahme ist korrekt, wenn die Wahrscheinlichkeit des Eintreffens eines zweiten Ereignisses, während der initialen Reaktion auf das vorangegangene Ereignis, sehr klein ist.
- Esterel erlaubt das gleichzeitige Auftreten von mehreren Eingabeereignissen. Die Reaktion ist in Esterel dann vollständig, wenn das System auf alle Ereignisse reagiert hat.

Determinismus

- Esterel ist deterministisch: auf eine Sequenz von Ereignissen (auch gleichzeitigen) muss immer dieselbe Sequenz von Ausgabe Ereignissen folgen.
- Alle Esterel-Anweisungen und -Konstrukte sind garantiert deterministisch. Die Forderung nach Determinismus wird durch den Esterel Compiler überprüft.
- Durch den Determinismus wird die Verifikation von Anwendungen wesentlich vereinfacht, allerdings birgt er auch die Gefahr, dass Ereignisse „vergessen“ werden, falls sie exakt zeitgleich mit höher priorisierten Ereignissen eintreffen.

Grundlagen der Esterel Sprache: Signale / Sensoren

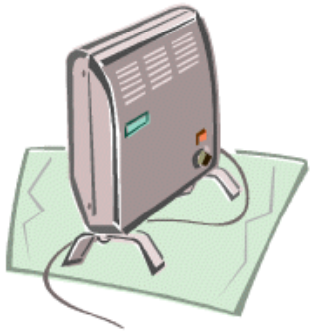
- Die Kommunikation im System erfolgt durch Signale und Sensoren
  - Sensoren stellen Messwerte zur Verfügung; Es muss immer ein Wert anliegen.
  - Signale können zur Ein- und zur Ausgabe verwendet werden. Es muss nicht immer ein Wert anliegen.
- Man unterscheidet dabei zwei Arten von Signalen:
  - Wertbehaftete Signale (Signal liegt mit einem Wert an oder liegt nicht an)
  - *pure* Signale (Signal liegt an oder nicht)
- Esterel Programme können in mehrere Module aufgeteilt werden
- Kommunikation der Module erfolgt über einen Broadcast-Mechanismus

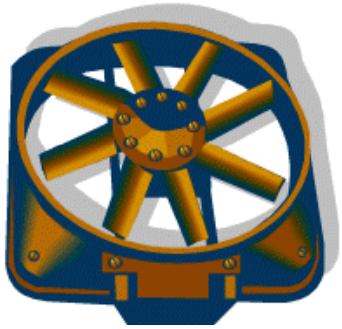


Grundlagen der Esterel Sprache: Programmsteuerung

- Das Verhalten des Programms wird durch Signale beeinflusst:
  - *emit*: Ein Signal wird ausgesandt.
  - *await*: Auf das Auftreten eines Signals wird gewartet.
  - *present*: Prüft, ob ein Signal anliegt.
- Die Ausführung eines Moduls kann mittels `abort` abgebrochen werden:
  - Syntax: `abort Body when Exit_Condition`
- Ein periodisches Ausführen von Anweisungen kann mit `every` umgesetzt werden.
  - Syntax: `every Occurrence do Body end every`
- Komposition von Anweisungen:
  - Blöcke von Befehlen (Module) können **nacheinander** oder **gleichzeitig** ausgeführt werden.
  - Blöcke von Befehlen (Module) können **wiederholt** ausgeführt werden.
  - Blöcke von Befehlen (Module) können **unterbrochen** werden.

Beispiel: Einfache Temperaturregelung





Beschreibung Beispiel

- Ziel: Regelung der Temperatur (Betriebstemperatur 5-40 Grad Celsius) mittels eines sehr einfachen Reglers.
- Ansatz:
  - Nähert sich die Temperatur einem der Grenzwerte, so wird der Lüfter bzw. die Heizung (Normalstufe) eingeschaltet.
  - Verbleibt der Wert dennoch im Grenzbereich, so wird auf die höchste Stufe geschaltet.
  - Ist der Wert wieder im Normalbereich, so wird (zur Vereinfachung) der Lüfter bzw. die Heizung wieder ausgeschaltet.
  - Wird die Betriebstemperatur über- bzw. unterschritten, so wird ein Abbruchssignal geschickt.

Esterel Code für Temperatur-Regelung (Auszug)

```
module TemperatureControler:
input TEMP: integer, SAMPLE_TIME, DELTA_T;
output HEATER_ON, HEATER_ON_STRONG,
       HEATER_OFF,VENTILATOR_ON, VENTILATOR_OFF,
       VENTILATOR_ON_STRONG, ABORT;

relation SAMPLE_TIME => TEMP;

  signal COLD, NORMAL, HOT in
    every SAMPLE_TIME do
      await immediate TEMP;
      if  ?TEMP<5 or ?TEMP>40 then emit ABORT
      elsif ?TEMP>=35 then emit HOT
      elseif ?TEMP<=10 then emit COLD
      else emit NORMAL
      end if
    end every
    ||
    loop
      await
        case COLD do
          emit HEATER_ON;
          abort
            await NORMAL;
            emit HEATER_OFF;
          when DELTA_T do
            emit HEATER_ON_STRONG;
            await NORMAL;
            emit HEATER_OFF;
          end abort
        case HOT do
%...
      end await
    end loop
  end signal
end module
```

Esterel-Konstrukt: Module

- **Module** definieren in Esterel (wiederverwendbaren) Code. Module haben ähnlich wie Unterprogramme ihre eigenen Daten und ihr eigenes Verhalten.
- Allerdings werden Module nicht aufgerufen, vielmehr findet eine Ersetzung des Aufrufs durch den Modulcode zur Übersetzungszeit statt (Inlining).
- Globale Variable werden nicht unterstützt. Ebenso sind rekursive Moduldefinitionen nicht erlaubt.
- Syntax:

```
%this is a line comment
module module-name:
declarations and compiler directives
%signals, local variables etc.
body
end module % end of module body
```

Esterel-Konstrukt: Parallele Komposition

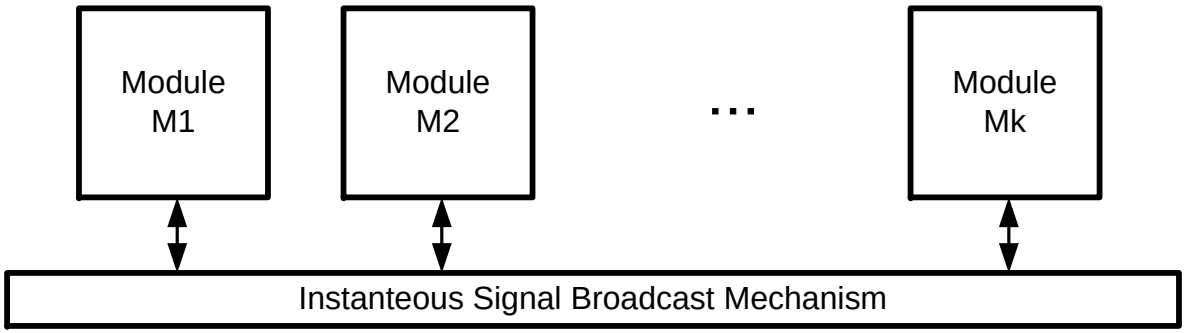
- Zur parallelen Komposition stellt Esterel den Operator || zur Verfügung. Sind P1 und P2 zwei Esterel-Programme, so ist auch P1||P2 ein Esterel-Programm mit folgenden Eigenschaften:
  - Alle Eingabeereignisse stehen sowohl P1 als auch P2 zur Verfügung.
  - Jede Ausgabe von P1 (oder P2) ist im gleichen Moment für P2 (oder P1) sichtbar.
  - Sowohl P1 als auch P2 werden parallel ausgeführt und die Anweisung P1||P2 endet erst, wenn beide Programme beendet sind.
  - Es können keine Daten oder Variablen von P1 und P2 gemeinsam genutzt werden.
- Zur graphischen Modellierung stehen parallele Teilautomaten zur Verfügung.

Signale

- Zur Kommunikation zwischen Komponenten (Modulen) werden Signale eingeführt. Signale sind eine logische Einheit zum Informationsaustausch und zur Interaktion.
- Die *Deklaration* eines Signals erfolgt am Beginn des Moduls. Der Signalname wird dabei typischerweise in Großbuchstaben geschrieben. Zudem muss der Signaltyp festgelegt werden.
- Esterel stellt verschiedene Signale zur Verfügung. Die Klassifikation erfolgt nach:
  - Sichtbarkeit: Schnittstellen (interface) Signale vs. lokale Signale
  - Enthaltener Information: pure Signale vs. wertbehaftete Signale (typisiert)
  - Zugreifbarkeit der Schnittstellensignale: Eingabe (input), Ausgabe(output), Ein- und Ausgabe (inputoutput), Sensor (Signal, das immer verfügbar ist und das nur über den Wert zugreifbar ist)

Esterel-Konstrukt: Broadcast-Mechanismus

- **Versand**: Der Versand von Signalen durch die `emit` Anweisung (terminiert sofort) erfolgt über einen Broadcast-Mechanismus, d.h. Signale sind immer sofort für alle anderen Module verfügbar.
  Die `sustain` Anweisung erzeugt in jeder Runde das entsprechende Signal und terminiert nicht.
- **Zugriff**: Prozesse können per `await` auf Signale warten oder prüfen, ob ein Signal momentan vorhanden ist (`if`). Auf den Wert eines wertbehafteten Signals kann mittels des Zugriffsoperator ? zugegriffen werden.



Esterel-Konstrukt: Ereignisse (Events)

- **Ereignisse** setzen sich zu einem bestimmten Zeitpunkt (**instant**) aus den Eingabesignalen aus der Umwelt und den Signalen, die durch das System als Reaktion ausgesendet werden, zusammen.
- Esterel-Programme können nicht direkt auf das ehemalige oder zukünftige Auftreten von Signalen zurückgreifen. Auch kann nicht auf einen ehemaligen oder zukünftigen Moment zugegriffen werden.
- Einzige Ausnahme ist der Zugriff auf den letzten Moment. Durch den Operator `pre` kann das Auftreten in der vorherigen Runde überprüft werden.

Beziehungen (relations)

- Der Esterel-Compiler erzeugt aus der Esterel-Datei einen endlichen Automaten. Hierzu müssen für jeden Zustand (Block) sämtliche Signalkombinationen getestet werden.
- Um bei der automatischen Generierung des endlichen Automaten des Systems die Größe zu reduzieren, können über die `relation` Anweisung Einschränkungen in Bezug auf die Signale spezifiziert werden:
- `relation Master-signal-name => Slave-signal-name;`

  Bei jedem Auftreten des Mastersignals muss auch das Slave-Signal verfügbar sein.
- `relation Signal-name-1 # Signal-name-2 # ... # Signal-name-n;`

  In jedem Moment darf maximal eines der spezifizierten Signale Signal-name-1, Signal-name-2 ,..., Signal-name-n präsent sein.

Zeitdauer

- Die Zeitachse wird in Esterel in diskrete Momente (**instants**) aufgeteilt. Über die Granularität wird dabei in Esterel keine Aussage getroffen.
- Zur deterministischen Vorhersage des zeitlichen Ablaufes von Programmen wird jede Anweisung in Esterel mit einer genauen Definition der Ausführungszeitdauer verknüpft.
- So terminiert beispielsweise `emit` sofort, während `await` so viel Zeit benötigt, bis das assoziierte Signal verfügbar ist.
- Auf den folgenden Folien werden die wichtigsten Konstrukte erläutert.

Esterel-Konstrukt: await Anweisung

```
await
    case Occurence-1 do Body-1
    case Occurence-2 do Body-2
    ...
    case Occurence-n do Body-n
end await;
```

- Mit Hilfe dieser Anweisung wird auf das Eintreten einer Bedingung gewartet. Im Falle eines Auftretens wird der assoziierte Code gestartet. Werden in einem Moment mehrere Bedingungen wahr, entscheidet die textuelle Reihenfolge. So kann eine deterministische Ausführung garantiert werden.

Esterel-Konstrukt: Unendliche Schleife (infinite loop)

```
loop Body end loop;
```

- Mit Hilfe dieser Anweisung wird ein Stück Code Body endlos ausgeführt. Sobald eine Ausführung des Codes beendet wird, wird der Code wieder neu gestartet.
- **Bedingung**: die Ausführung des Codes darf nicht im gleichen Moment, indem sie gestartet wurde, terminieren.

Esterel-Konstrukt: abort

- Zur einfacheren Modellierung können Abbruchbedingungen nicht nur durch Zustandsübergänge, sondern auch direkt mit Makrozuständen verbunden werden.
- Dabei wird zwischen zwei Arten des Abbruches unterschieden:
  - weak abort: die in der Runde vorhandenen Signale werden noch verarbeitet, danach jedoch der Abbruch vollzogen
  - strong abort: der Abbruch wird sofort vollzogen, eventuell vorhandene Signale ignoriert.
- In der Sprache Esterel wird eine Abbruchbedingung durch das Konstrukt
  `abort Body when Exit_Condition`
  bzw.
  `abort Body when immediate Exit_Condition`
  ausgedrückt.

Esterel-Konstrukt: Lokale und wertbehaftete Signale

```
signal Signal-decl-1,Signal-decl-
2,...,Signal-decl-n in
   Body
end;
```

- Durch diese Anweisung werden lokale Signale erzeugt, die nur innerhalb des mit Body bezeichneten Code verfügbar sind.

```
Signal-name: Signal-type
```

- Der Typ eines wertbehaften Signals kann durch diese Konstruktion spezifiziert werden.

Esterel-Konstrukt: every Anweisung

- Mit Hilfe der `every` Anweisung kann ein periodisches Wiederstarten implementiert werden.
- Syntax:

```
every Occurence do
    Body
end every
```

- Semantik: Jedes Mal falls die Bedingung `Occurence` erfüllt ist, wird der Code Body gestartet. Falls die nächste Bedingung `Occurence` vor der Beendigung der Ausführung von Body auftritt, wird die aktuelle Ausführung sofort beendet und eine neue Ausführung gestartet.
- Es ist auch möglich eine Aktion in jedem Moment zu starten:

```
every Tick do
    Body
end every;
```

Esterel-Konstrukt: if Anweisung in Bezug auf Signale

- Durch Verwendung der if- Anweisung kann auch die Existenz eines Signals geprüft werden.
- **Syntax**:

```
if Signal-Name then
    Body-1
else
    Body-2
```

- **Semantik**: Bei Start dieser Anweisung wird geprüft, ob das Signal `Signal-Name` verfügbar ist. Ist es verfügbar, so wird der Code von `Body-1` ausgeführt, anderenfalls von Body-2. Innerhalb der Anweisung `if` kann auch entweder der `then Body-1` oder der `else Body-2`-Teil weggelassen werden.

Automaten zur Modellierung von reaktiven Systemen

Automaten im Kontext von reaktiven Systemen

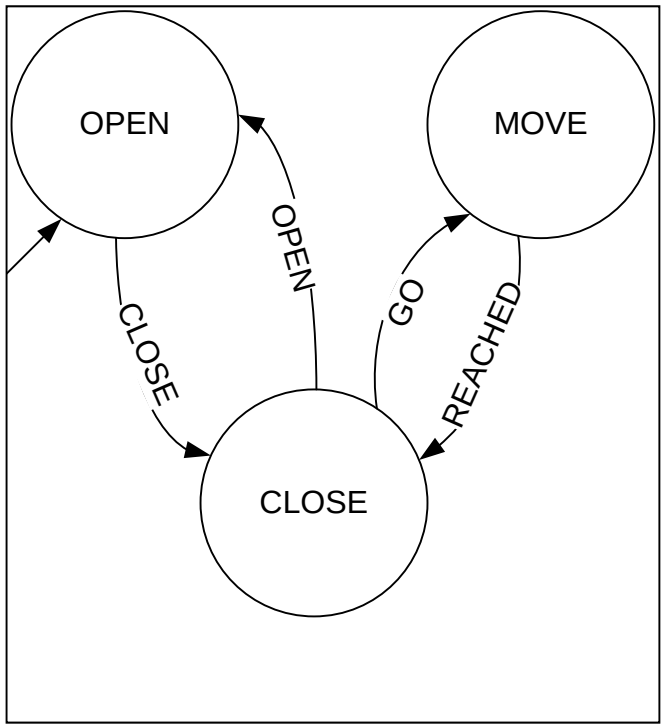
- Durch graphische Darstellung (z.B. Automaten) kann die Verständlichkeit des Codes stark verbessert werden.

- Ein reaktives System kann durch die zyklische Ausführung folgender Schritte beschrieben werden:
  1. Lesen der Eingangssignale
  2. Berechnung der Reaktionen
  3. Auslösen der Ausgangssignale

- Die zyklische Ausführung kann im Automatenmodell wie folgt interpretiert werden:
  1. Lesen der Eingabe im aktuellen Zustand
  2. Berechnen der Zustandsübergangsfunktion und ggfs. Zustandswechsel
  3. Erzeugung von Ausgangssignalen (abhängig von alten Zustand und gelesenem Eingangssignal)

- Die Synchronitätshypothese bedeutet im Bezug auf den Automaten, dass im Vergleich zur Zeit für Änderungen der Umgebung eine vernachlässigbare Zeit für die Berechnung der Zustandsübergange und Ausgabefunktion benötigt wird.

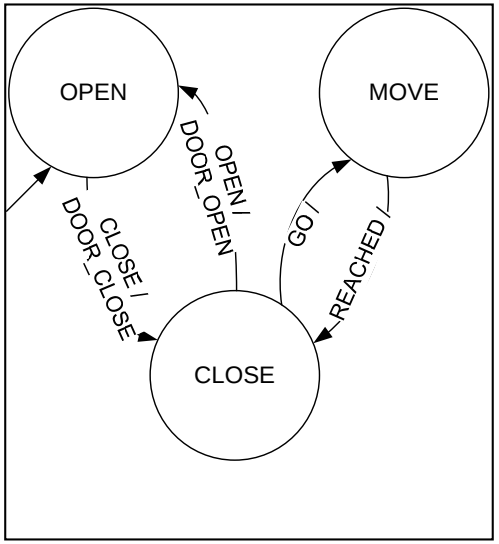
Endliche Automaten

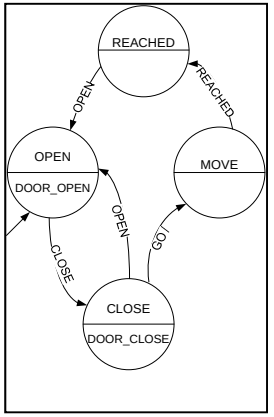


- Ein klassischer endlicher Automat (Q, $\sum$, δ, S, F) ist eine endliche Menge von Zuständen und Zustandsübergängen mit:
  - endlicher Menge von Zuständen Q
  - endliches Eingabealphabet $\sum$
  - Übergangsfunktion δ : Q x $\sum \rightarrow$ Q
  - Endlicher Menge von Startzuständen S
  - Menge von Endzuständen F $\subseteq$ Q
- Um die Verständlichkeit zu verbessern, werden nur deterministische Automaten modelliert, also |S|=1 und $\delta(q,\alpha)=q' \wedge \delta(q,\alpha)=q'' \Rightarrow q'=q''$
- Problem bei der klassischen Definition von Automaten: Ausgaben können nicht modelliert werden.

Automaten mit Ausgaben

- Mealy- und Moore-Automaten unterstützen die Ausgabe von Signalen
- Die Ausgaben von Mealy-Automaten ($Q,\Sigma,\Omega, \delta, \lambda, S,F$) sind dabei an die Übergänge gebunden mit
  - Q, Σ, δ, S, F wie bei klassischem Automat
  - Ausgabealphabet Ω
  - Ausgabefunktion $\lambda: Q \times \Sigma \rightarrow \Omega$
- Bei Moore-Automaten ($Q,\Sigma,\Omega, \delta, \lambda, S,F$) ist die Ausgabe dagegen von den Zuständen abhängig:
  - $Q, \Sigma, \Omega, \delta, S, F$ wie bei Mealy-Automat
  - Ausgabefunktion $\lambda: Q \rightarrow \Omega$
- Moore- und Mealy-Automat sind gleich mächtig: sie können ineinander konvertiert werden.





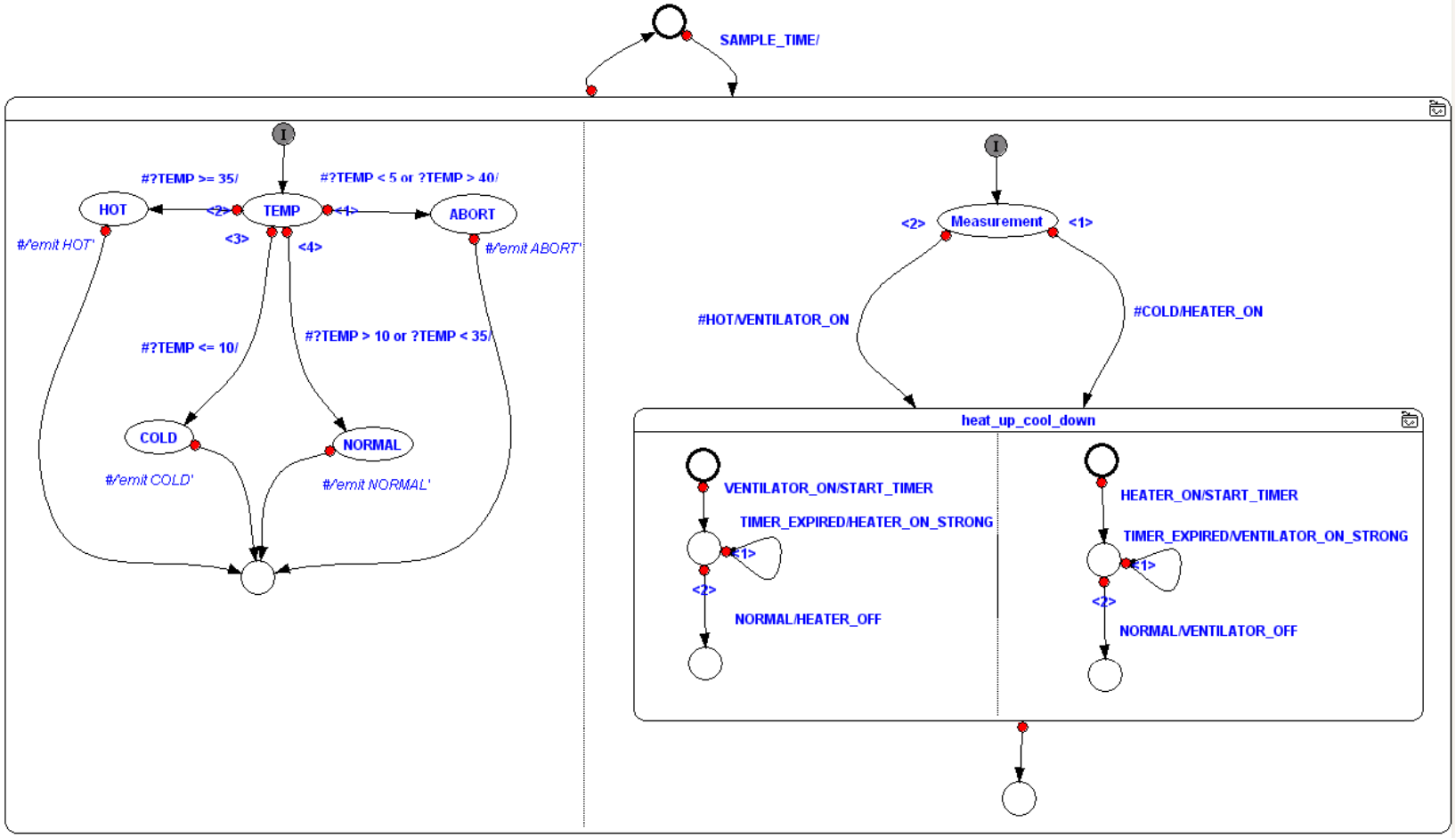
Harel-Automaten / Statecharts

- Heutiger Standard zur Beschreibung von reaktiven Systemen sind die von David Harel 1987 vorgeschlagenen Statecharts.
- Statecharts zeichnen sich durch folgende Eigenschaften aus:
  - Vereinigung der Eigenschaften von Mealy- und Moore-Automaten
    - Ausgaben in Abhängigkeit von Zustandsübergängen möglich
    - Durchführung von Ausgaben beim Erreichen eines Zustands (**`onEntry`**) oder Verlassen eines Zustandes (**`onExit`**)
  - Zur Erhöhung der Lesbarkeit: Hierarchische Strukturierung von Teilautomaten möglich inkl. Gedächtnisfunktionalität (History)
  - Darstellung paralleler Abläufe durch parallele Teilautomaten.
  - Verknüpfung von Zuständen mit Aktionen: Befehle **`do`** (zeitlich begrenzte Aktivität), **`throughout`** (zeitlich unbegrenzte Aktivität)
  - Einführung spontaner bzw. überwachter (guarded) Übergänge

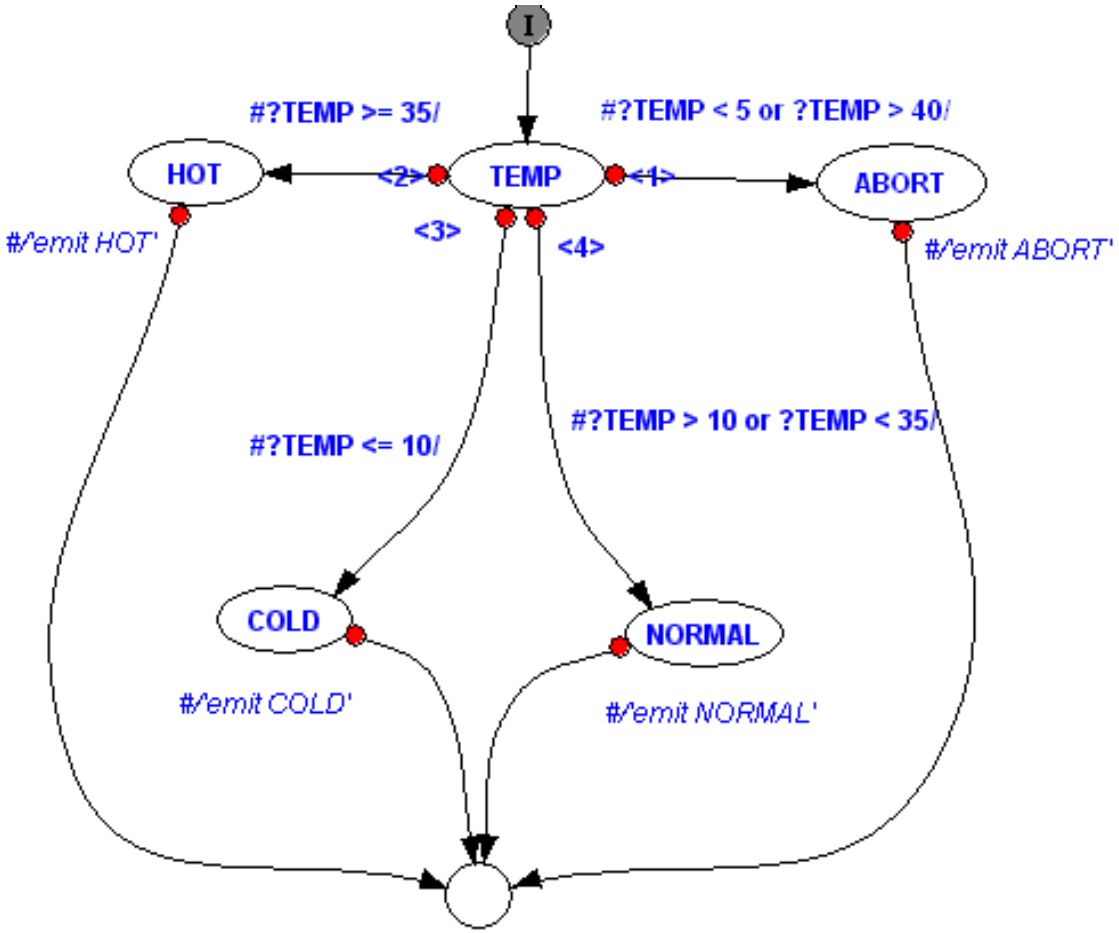
Safe State Machine

- Esterel benutzt eine eigene Klasse von Automaten, die den Statecharts sehr ähnlich sind:
  - Vereinigung der Eigenschaften von Mealy- und Moore-Automaten
    - Ausgaben in Abhängigkeit von Zustandsübergängen möglich
    - Durchführung von Ausgaben beim Erreichen eines Zustands (**`onExit`**) oder Verlassen eines Zustandes (**`onEntry`**)
  - Zur Erhöhung der Lesbarkeit: Hierarchische Strukturierung von Teilautomaten möglich inkl. Gedächtnisfunktionalität
  - Darstellung paralleler Abläufe durch parallele Teilautomaten.
  - ~~Verknüpfung von Zuständen mit Aktionen: Befehle **`do`** (zeitlich begrenzte Aktivität), **`throughout`** (zeitlich unbegrenzte Aktivität)~~
  - Einführung ~~spontaner~~ bzw. überwachter (guarded) Übergänge
  - Zusätzliche Esterel abhängige Konstrukte (z.B. `pre` Operator)

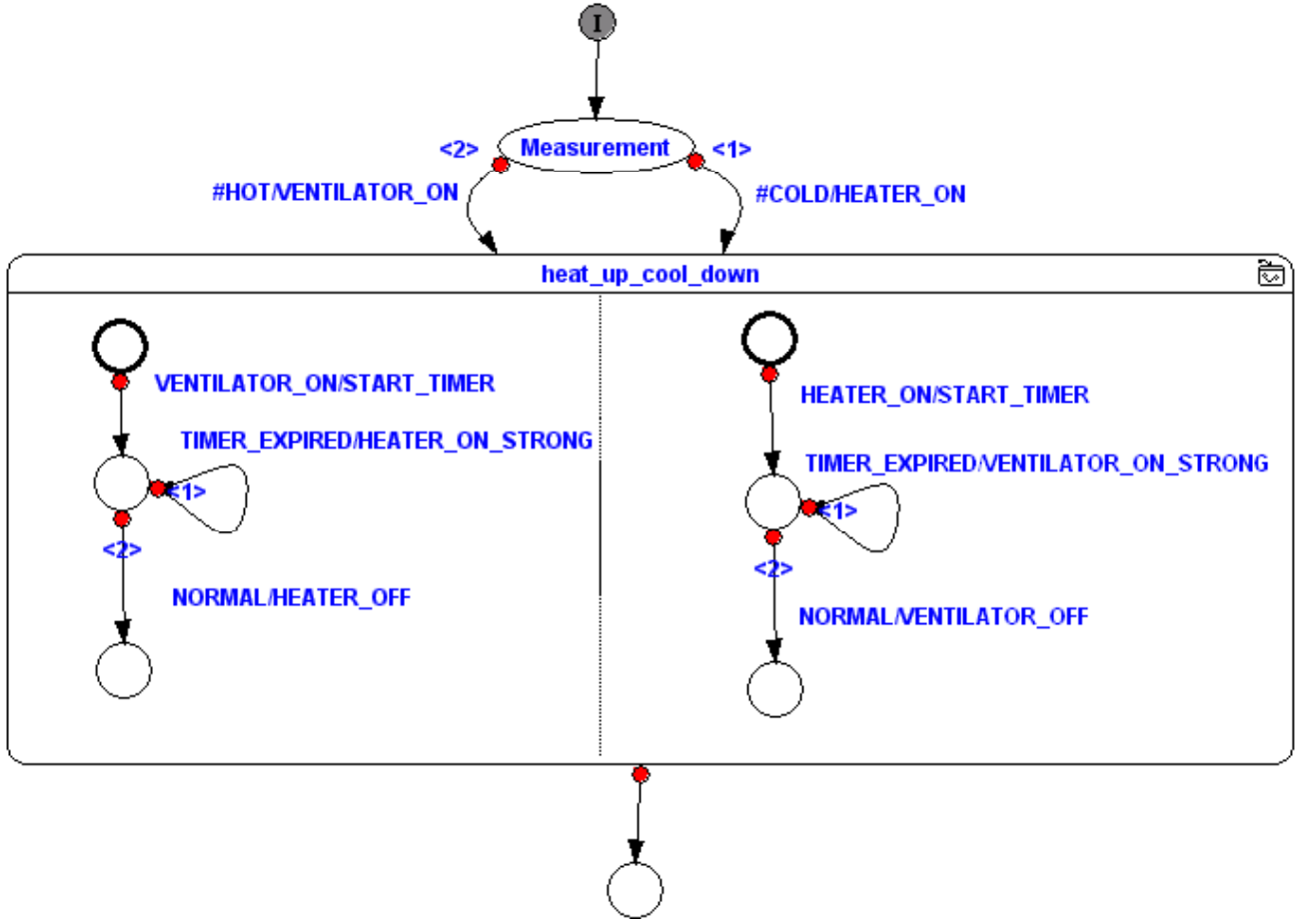
Beispiel als Automat



Beispiel als Automat – Teil 1



Beispiel als Automat – Teil 2



Modellierung von Echtzeitsystemen

Verifikation von Echtzeitsystemen - Einsatz von Formalen Methoden

Problemstellung

„As soon as we started programming, we found to our surprise that it wasn‘t as easy to get programs right as we had thought.
Debugging had to be discovered.
I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.“

Maurice Wilkes.
(Turing Award 1967)

Verifikation & Validierung

- Verifikation: Um die Korrektheit von Programmen in Bezug auf die Spezifikation zu garantieren, wird eine formale Verifikation benutzt. Dazu werden mathematische Korrektheitsbeweise durchgeführt.
- Validierung: Durch eine Validierung kann überprüft werden, dass das System als Modell hinreichend genau nachgebildet wird.
  Techniken:
  - Inspektion
  - Plausibilitätsprüfung
  - Vergleich unabhängig entwickelter Modelle
  - Vergleichsmessung an einem Referenzobjekt

Übersicht über (formale) Methoden

- Deduktive (SW-)Verifikation
  - Beweissysteme, Theorembeweiser
- Model Checking
  - für Systeme mit endlichem Zustandsraum
  - Anforderungsspezifikation mit temporaler Logik
- Testen
  - spielt in der Praxis eine große Rolle
  - sollte systematisch erfolgen $\rightarrow$ ausgereifte Methodik
  - ... stets unvollständig