Die synchrone, imperative Sprache

Design, Semantik und Implementierung

ESTEREL

Seminar Synchrone Programmiersprachen

```
module FOO :
input S1 (integer);
output S2 (integer), S3 (Integer);
loop
   signal S (integer) in
      emit S(0);
      await S1;
      emit S(1);
   ||
      emit S2(?S);
      await S;
      emit S3(?S);
   end
end
```

Samuel Eickelberg

Johannes Dahlke

Agenda


Beispiel
Semantiken
Plain-Esterel
Basic-Esterel
Grundlagen
Einführung

Einführung

- Esterel – kurz gefasst
- Reaktive Systeme
- Entwicklungswerkzeuge
- Synchronitätshypothese

Esterel – Ein historischer Abriss

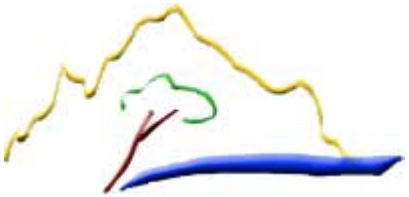
Wann

- Anfänge in 1982

Wer

- Team der Mines ParisTech und INRIA:

  Marmorat, Rigault, Berry, Moisan, Gonthier, Cosserat, ...



Name

- Esterel Gebirge (Côte d’Azur)

Verwandt

- Lustre, Signal

Esterel – Ein historischer Abriss

Compiler

- Brzozowski-Algorithmus
  - → Automaten
    - → Zustandsexplosionen
- Hardwarecompiler

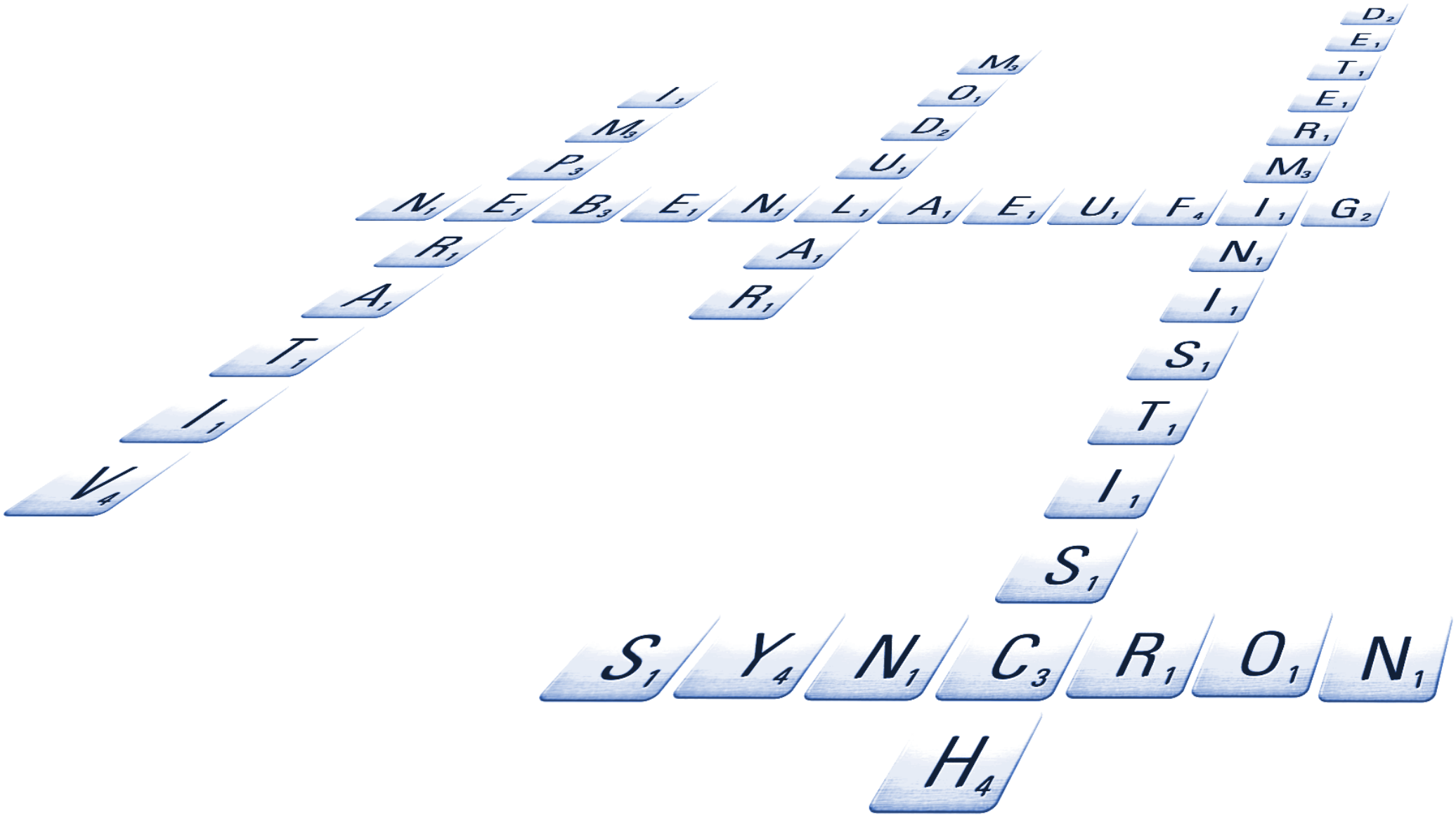
Konzept

- Modulare Programmierung

Produkt

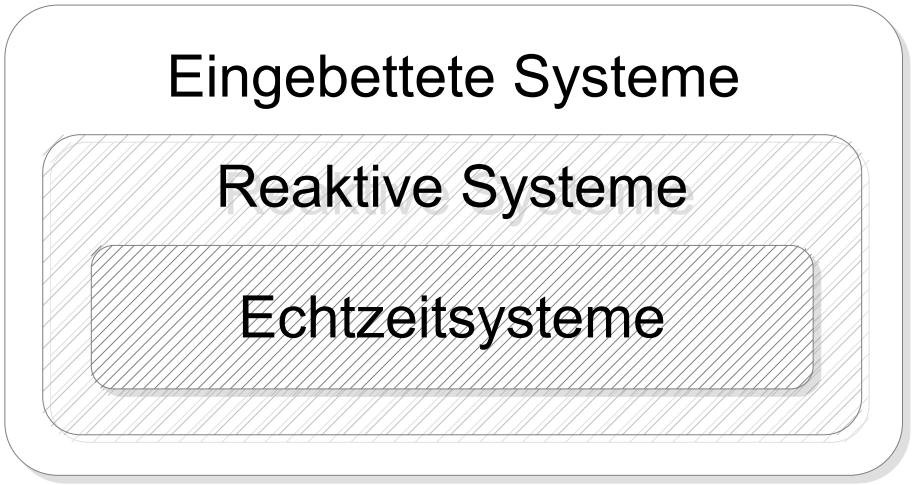
- Esterel Studio ™

Esterel – Eigenschaften



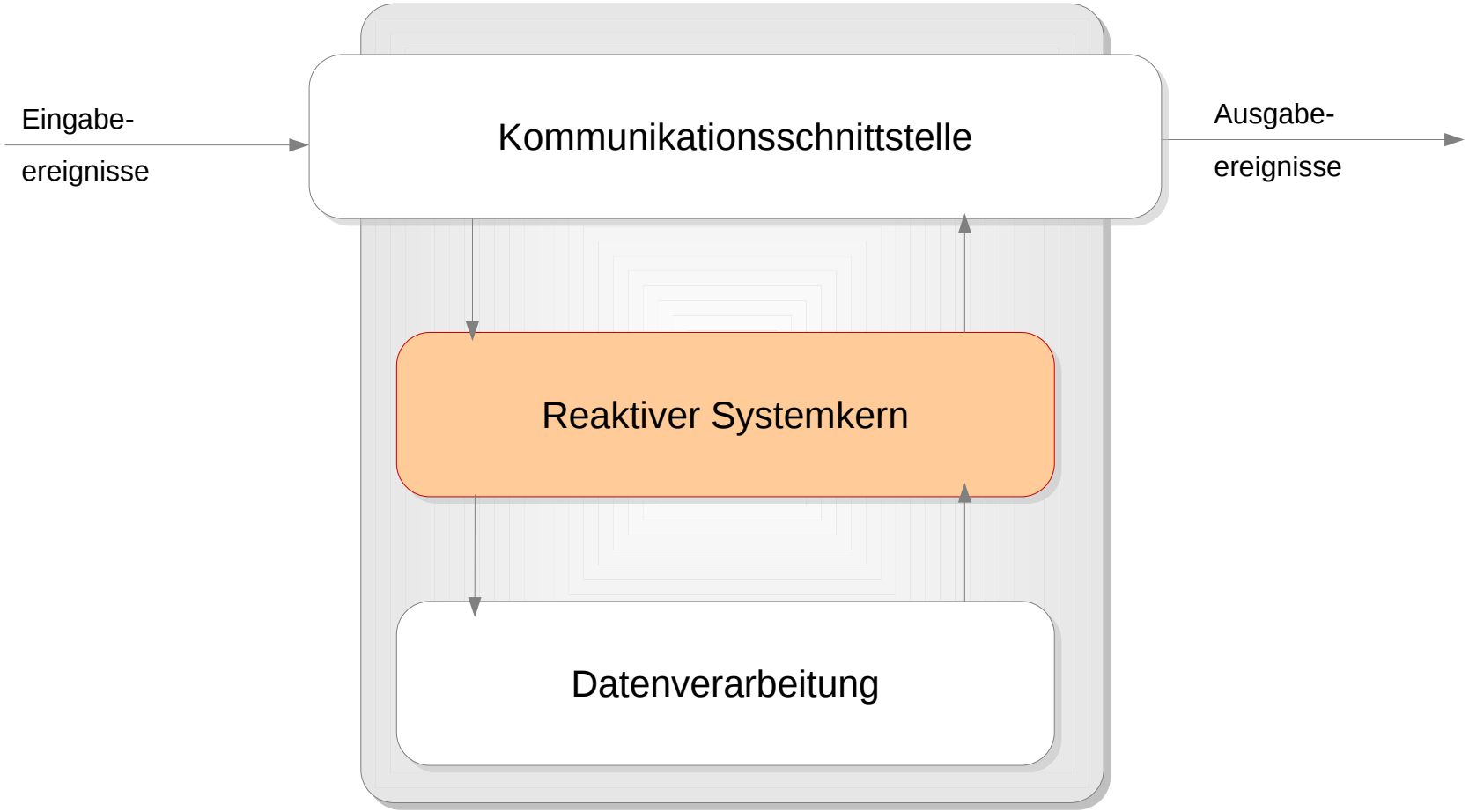
Reaktive Systeme

Einordnung



eingabegesteuert

Reaktive Systeme



Einsatz reaktiver Systeme

- Automotive
- Avionik
- Kernkraftwerke
- Medizintechnik

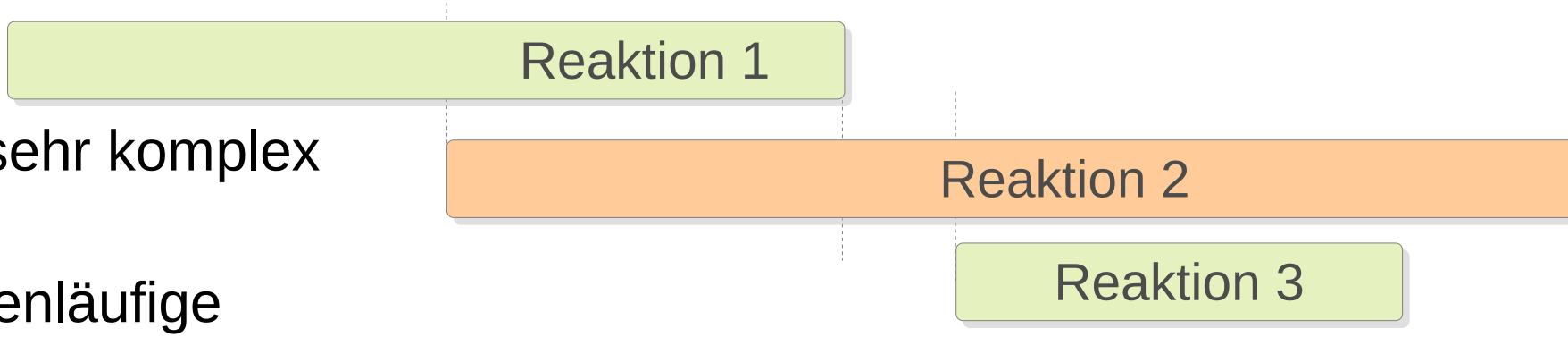
Entwicklungswerkzeuge

Werkzeuge zur Entwicklung reaktiver Systeme

- Automaten
- Petrinetze
- Asynchrone Programmiersprachen
- Synchrone Programmiersprachen

Determinismus und Nebenläufigkeit

Reaktive Systeme



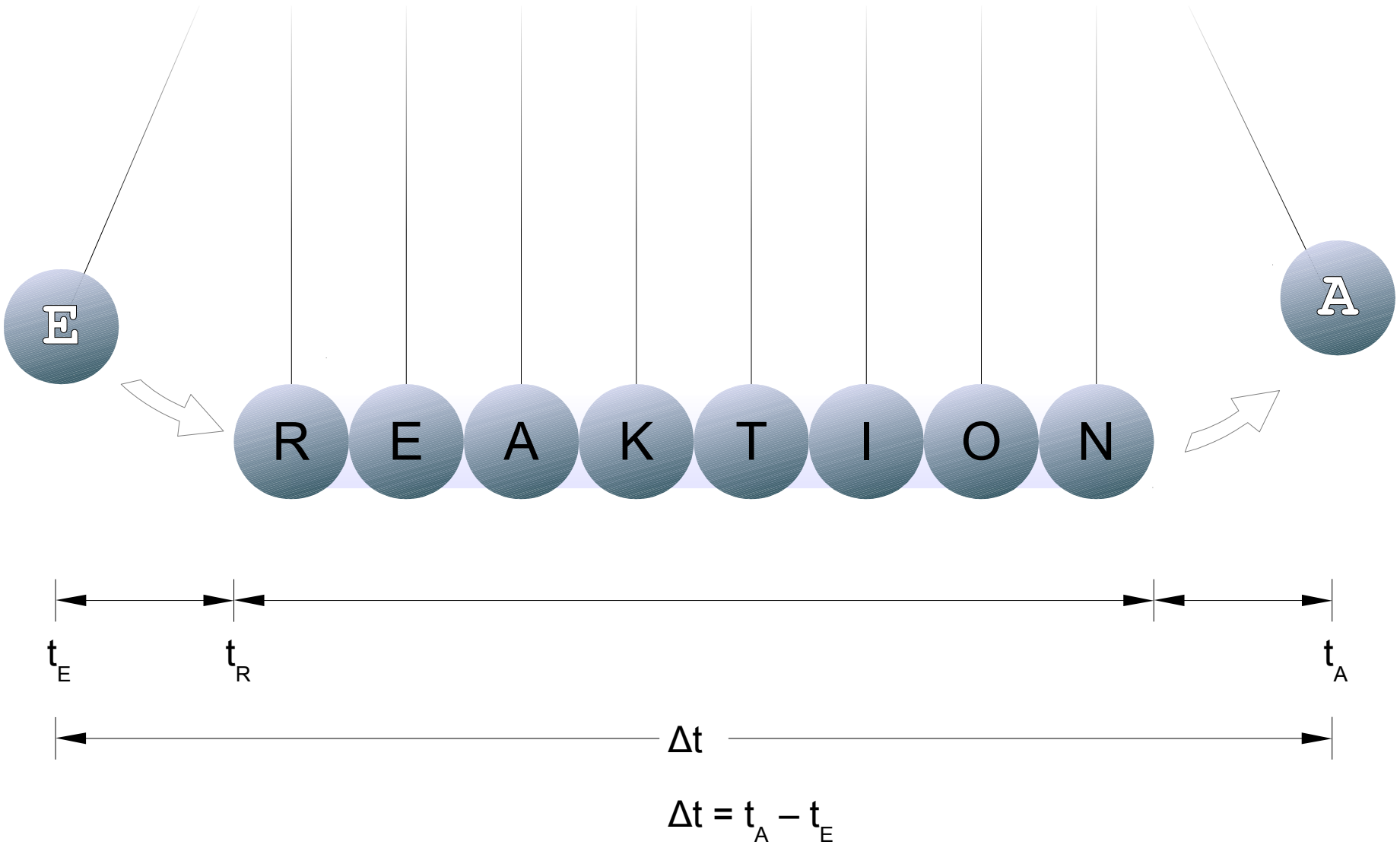
- oft sehr komplex
- nebenläufige Teilkomponenten
- Kooperation → Nichtdeterminismus ?
- deterministisch, wenn

  bestimmte Eingabesequenz

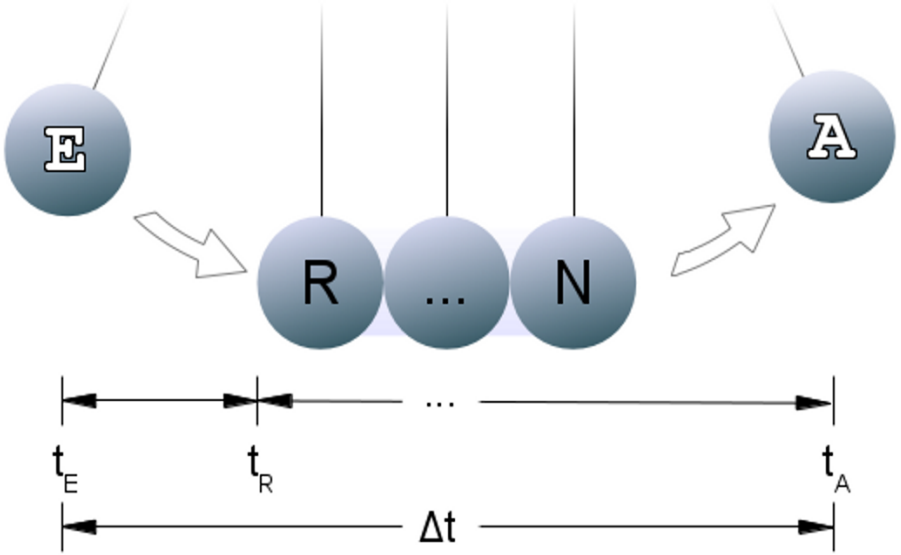
  ⇒

  stets gleiche Ausgabesequenz

Synchronitätshypothese



Synchronitätshypothese



- Eine Reaktion erfolgt **unverzüglich** (*zero delay*)

$t_E = t_R$

- Eine Reaktion ist **unteilbar** (*atomar*)

- Eine Reaktion geschieht

$\Delta t = 0$ — **augenblicklich** (*control takes no time*)

$\Delta t = t_A - t_E$

$\Delta t > 0$ — Ausnahmen: Trigger, Watchdog, Delay, ...

Synchronitätshypothese

- Befehle kosten Zeit

  ⇔

  Zeitkonsum ihr Zweck ist

- Annahme:
  Taktrate des Rechners ist **unendlich**

- Theoretisches Modell


Kommunikationsschnittstelle
augenblicklich
Entwicklung
unverzüglich
synchron
Reaktive Systeme
deterministisch
Petrinetze
reaktiv
sofortig
reaktiver Systemkern
Synchronitätshypothese
Automaten
unteilbar
atomar
sychrone Sprachen
eingabegesteuert
unendliche Taktrate
Esterel
Hostsprache
Werkzeuge
asynchrone Sprachen
imperativ
nebenläufig
Reaktion
modular
Datenverarbeitungsebene

Vorankommen

Esterel – Grundlagen

- Modul
- Parallelität
- Datendeklarationen
- Datenverarbeitungsebene
- Kommunikationsschnittstelle
- Relationen

Modul

Modularisierung

- Esterelprogramm := mindestens 1 Modul
- Modul := Selbstständiges Programmfragment
- Kein globaler Geltungsbereich

Wiederverwendung

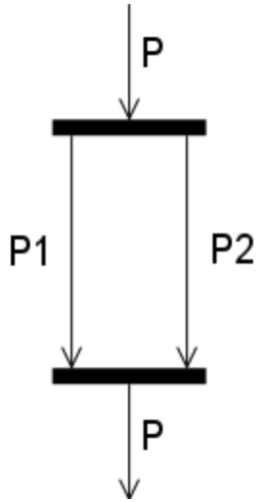
- Direktive `copymodule M`
- Ähnlich C-Makros

Parallelität

- Operator ||

- Seien P1 und P2 Programme.

  Dann ist auch P1 || P2 ein Programm.



- Die parallele Ausführung von P1 || P2 terminiert, wenn sowohl P1 als auch P2 terminieren.

Parallelität

- Ausgaben sind öffentlich.

```
emit S(1); || emit S2(?S);
```

Signal S2 wird mit Wert 1 emmitiert.

- P1 und P2 können sich keine Variablen teilen.

Datendeklarationen

- Primitive Datentypen: integer, boolean, triv, float, string
- ~~Verbundtypen~~ und ~~zusammengesetzte Datentypen~~
- Deklaration benutzerdefinierter Typen

```
type DOUBLE, TIME;
```

  Definition der Typen in Hostsprache

- Konstantendeklaration

```
constant PI : DOUBLE, NOON : TIME;
```

  Wertzuweisung in Hostsprache

Datendeklarationen

- Deklaration einer Funktion

```
function SQRT(DOUBLE) : DOUBLE;
function EQUALS(TIME,TIME) : boolean;
```

Annahme: seiteneffektfrei

- Deklaration einer Prozedur

```
procedure INC_TIME(TIME)(integer);
```

Datenverarbeitungsebene

- Datentyp `TIME` in Hostsprache C

```
typedef struct {
   int hours;
   int minutes;
   int seconds;
} TIME;
```

- Vergleichsoperator für `TIME` in C

```
int _eq_TIME ( t1, t2)
   TIME t1, t2;    //  old-style syntax
{
   return (     t1.hours == t2.hours
             && t1.minutes == t2.minutes
             && t1.seconds == t2.seconds);
}
```

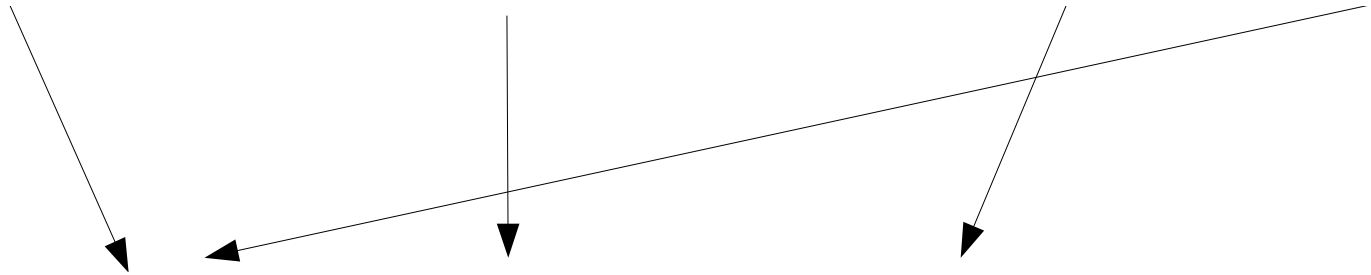
Datenverarbeitungsebene

```
procedure _text_to_TIME (TIME, string) ();
```

```
void _text_to_TIME (time_ptr, str)
  TIME* time_ptr;
  char* str;
{
  sscanf( str, "%d:%d:%d",
          &(time_ptr->hours),
          &(time_ptr->minutes),
          &(time_ptr->seconds));
}
```

Datenverarbeitungsebene

```
function _TIME_to_text () (time) : TIME;
```



```
char* _TIME_to_text (time)
  TIME time;
{
  static char buf[9]="";
  sprintf( buf, "%02d:%02d:%02d",
           time_ptr->hours,
           time_ptr->minutes,
           time_ptr->seconds);
  return (buf);
}
```

Kommunikationsschnittstelle

Schnittstellen zur Außenwelt

- Signale (aktiv)
- Sensoren (passiv)

Klassifikation

- Zugriff
- Informationsgehalt
- Sichtbarkeit

Kommunikationsschnittstelle

Zugriff

- Nur-Eingabe-Signale
- Nur-Ausgabe-Signale
- Ein-und-Ausgabe-Signale
- Nur-Lese-Signale (Sensoren)

Sichtbarkeit

- Schnittstellensignale
- lokale Signale

Kommunikationsschnittstelle

Informationsgehalt

- wertelose Signale `S(v)`
- wertebehaftete Signale `S`

Signalzustand

entweder präsent oder nicht.

Signalstatus und Signalwert über die gesamte Reaktion fix.

Kommunikationsschnittstelle

Eingabesignale

- externe Ereignisse: Interrupts, Timer, Tastendrücke, …
- augenblickliche Verarbeitung
- Lesen der Werte mit Abfrageoperator `?`

  `?S = v` falls wertebehaftet

  `?S = triv` falls wertefrei

- Deklaration `input S (t)`

Kommunikationsschnittstelle

Ausgabesignale

- augenblicklich ausgelöst — `emit S(exp)`
- wertebehaftet `S(exp)` oder wertfrei `S`
- Deklaration — `output O (combine t with c)`

Kommunikationsschnittstelle

Kombinationsfunktion c

- Signatur `function c(t,t):t`
- Notwendig bei Mehrfach-Emission

```
emit S(1)||emit S(2)||...||emit S(n)
```

- Berechnet eindeutigen Ausgabewert

$$c(v_1, c(v_2, \ldots c(v_{n-2}, c(v_{n-1}, v_n)) \ldots))$$

Kommunikationsschnittstelle

Ein-und-Ausgabe-Signale

- Kombinierter Signaltyp
- Deklaration

```
inputoutput BUS_REQUEST
```

Sensor

- Für passive Geräte z.B. Thermometer
- Nur Abfrage per ?-Operator
- Deklaration

```
sensor TEMPERATURE (CELSIUS)
```

Kommunikationsschnittstelle

Lokale Signale

- Verlassen nicht die Kernelebene
- Eingeschränkter Geltungsbereich
- Deklaration `signal S (type) in ... end`
- Beispiel

```
signal S (integer) in % local signal
  emit S(0);
  await S1;
  emit S(1);
||
  emit S2(?S);
  await S;
  emit S3(?S);
end % end of scope of S
```

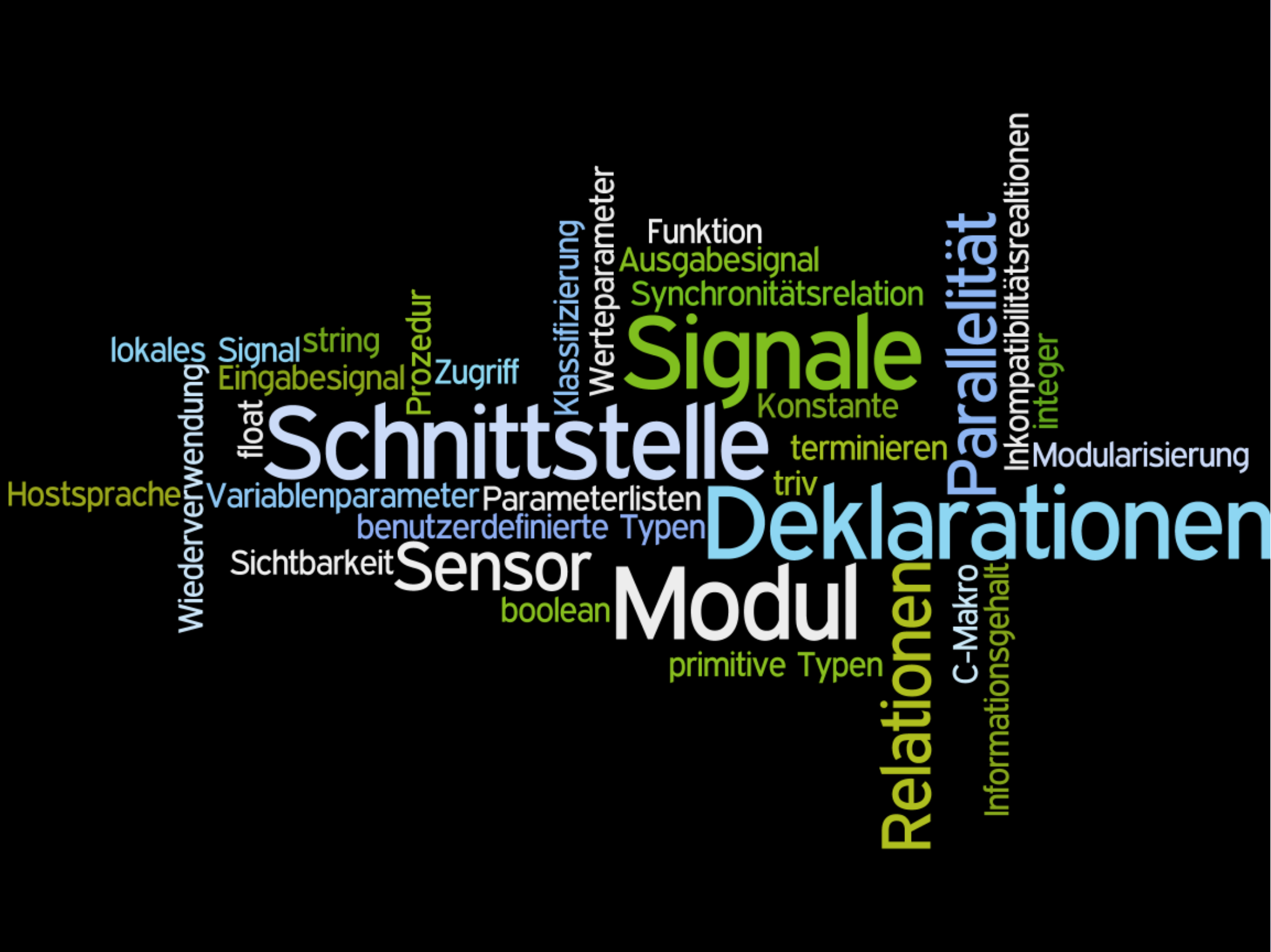
Relationen

- Inkompatibilitätsrelation

```
LEFT_BUTTON # RIGHT_BUTTON
```

- Synchronitätsrelation

```
SECOND => HUNDREDTH_OF_SECONDS
```


Funktion
Ausgabesignal
Synchronitätsrelation
Signale
Konstante
terminieren
triv
Deklarationen
Parallelität
Inkompatibilitätsrealtionen
integer
Modularisierung
Relationen
C-Makro
Informationsgehalt
primitive Typen
Modul
boolean
Sensor
Sichtbarkeit
benutzerdefinierte Typen
Parameterlisten
Variablenparameter
Hostsprache
Schnittstelle
float
Wiederverwendung
lokales Signal
string
Eingabesignal
Prozedur
Zugriff
Klassifizierung
Werteparameter

Vorankommen

Basic ESTEREL

- **ESTEREL** ist eine *imperative* Sprache
- Viele Sprachkonstrukte bereits intuitiv **vertraut**
- **Einiges ist allerdings neu und anders**

Basic ESTEREL

```
do
  every STEP do
    emit JUMP
  end
watching 100 METER
```

- Die innere Schleife wäre endlos, würde die äußere nicht die Abbruchbedingung enthalten

Intuitive Verhaltenssemantik

- beschreibt Ausführungsverhalten oberflächlich
  - Ausführungsmoment
  - Wann terminiert eine Anweisung?
- Orientierung am zeitlichen Ablauf
- Programm = Folge an Reaktionen
- Reaktion: augenblicklich und atomar

Sharing Law

- Grundsatz 1: Während einer Reaktion hat ein Signal einen konstanten *Status* (es ist präsent oder nicht).
- Grundsatz 2: Während einer Reaktion hat ein Signal einen konstanten *Wert*, selbst dann, wenn es nicht aktiv ist.
- Folgerung 1 aus den beiden Grundsätzen: Ein Signal, das nicht aktiv ist, hat immer den Wert, den es bei der *letzten* Reaktion hatte.
- Folgerung 2 aus den beiden Grundsätzen: Ist ein Signal nie aktiv gewesen, ist sein Wert *undefiniert*.

'Basic' ESTEREL: Befehlssatz (I)

```
% Dummy-Anweisung (NOP): Bewirkt nichts
nothing

% Halte-Anweisung: Erzwingt das Ende
% der Programmausführung
halt


% Zuweisung
X := exp

% Externer Routinenaufruf
call P (variable list) (expression list)

% Emission des Signals S mit dem Wert,
% den der Ausdruck exp liefert
emit S(exp)

% Anweisungssequenz
stat1 ; stat2


% Endlosschleife
loop
  stat
end

% Bedingte Verzweigung
if exp
  then stat1
  else stat2
end
```

- Basis für den erweiterten 'Plain' **ESTEREL**-Befehlssatz
- Vorsicht bei Endlosschleife: Widerspruch zur Synchronitäts-hypothese!
  (und daher nicht zulässig ohne Abbruchbedingung!)
- Metavariablen:
  - *type* (Typen)
  - *exp* (Ausdrücke)
  - *stat* (Anweisungen)
- Vieles ist (intuitiv) bekannt aus anderen imperativen Sprachen

'Basic' ESTEREL: Befehlssatz (II)

```
% Test auf Vorhandensein des Signals S
% mit bedingter Ergebnisbehandlung
present S
  then stat1
  else stat2
end

% Watchdog: Führt stat solange
% wiederholend aus, bis das Signal S
% empfangen wird.
do stat watching S

% Parallelanweisung
stat1 || stat2

% Definition der Trap-Marke T
trap T in
  stat
end

% Austritt aus der Trap mit Marke T
exit T

% Lokale Variablendeklaration
var X : type in
  stat
end

% Lokale Signaldeklaration
signal S (combine type with comb) in
  stat
end
```

- Es gibt 2 Arten bedingter Anweisungen:
  if-then-else und *present-then-else*
- **ESTEREL**-eigen sind:
  - `emit`
  - `present`
  - `watching`
  - `signal`
- *comb* steht für eine Kombinationsfunktion (für >=2 Signale)

'Basic' ESTEREL: Traps (Ausnahmen)

- Traps sind in anderen Sprachen, wie Java, vergleichbar mit Exceptions
- Das Java-Codeschnipsel unten verdeutlicht die Parallele zu **ESTEREL** (oben)
- Es werden 2 Ausnahmen gefangen und mit Handlern wird jeweils definiert, was dann zu tun ist

```
trap ALARM (combine integer with +),
     ZERO_DIVIDE, TERMINATE in
  stat
handle ALARM do stat1
handle ZERO_DIVIDE do stat2
end
```

```
try {
      stat;
} catch (ALARM_Exception e) {
      stat1;
} catch (ZERO_DIVIDE_Exception e) {
      stat2;
}
```

Beispiel für unerreichbaren Code

```
trap T1 in
  trap T2 in
      X := 0
    ||
      Y := 0; exit T2
    ||
      Z := 0; exit T1; Z := 1
  end;
  U := 0
end
```

- `X := 0, Y := 0` und `Z := 0` werden simultan ausgeführt
- Da nach `Z := 0` schon `T1` verlassen wird, kann `Z := 1` nicht ausgeführt werden
- `exit T2` wird verworfen, da `T1` `T2` umschließt
- `U := 0` wird nicht ausgeführt, da `T1` bereits verlassen wurde
- Endzustand:
  `X := 0, Y := 0, Z := 0`

Bsp. für unterschiedliche Scopes bei Signalen

```
module FOO :
input S1 (integer);
output S2 (integer), S3 (Integer);
loop
  signal S (integer) in % local signal
    emit S(0);
    await S1;
    emit S(1);
  ||
    emit S2(?S);
    await S;
    emit S3(?S);
  end % end of scope of S
end
```

- Im Modul `FOO` stehen 2 gleichzeitige Emissionen des lokalen Signals `S` nicht im gleichen Kontext
  (also braucht man hier keine Kombinationsfunktion!)
- Start der Scheife: `S` wird mit Wert 0 emittiert und es wird auf `S1` gewartet, dann `S` mit 1 emittiert
- Parallel dazu wird `S2` mit dem Wert von `S` emittiert und auf `S` gewartet, dann `S2` mit dem Wert von `S` emittiert

Vorankommen

Plain-Esterel

Erweiterung von Basic-Esterel

- Kommunikationsschnittstelle
- Syntaktischer Zucker
- Ausnahmebehandlung
- Modularität

Plain-Esterel

Schnittstelle ✓

- Ein-und-Ausgabe-Signal
- Sensor

Modularität ✓

- Direktive `copymodule M`

Plain-Esterel

Syntaktischer Zucker

- Abwarten
- kopfgesteuerte Schleife
- Timeout für Watchdogs
- Ereigniszähler
- Immediate-Watchdog

Syntaxerweiterungen

einfaches Abwarten

- gesüßt

```
await S
```

- ungesüßt

```
do
   halt
watching S
```

mehrfaches Abwarten

```
await
   case SECOND do stat1
   case 2 METER do stat2
   case immediate ALARM do stat3
end
```

Syntaxerweiterungen

kopfgesteuerte Schleife

- gesüßt

```
repeat exp times
   stat
end
```

- ungesüßt

```
trap T in
   X := 0;
   loop
      if X = exp
         then exit T
         else stat;
      X := X +1;
   end
end
```

Syntaxerweiterungen

Timeout für Watchdogs

- gesüßt

```
do
   stat1
watching S
timeout stat2
end
```

- ungesüßt

```
trap TERMINATE in
   do
      stat1 ;
      exit TERMINATE
   watching S
   stat2
end
```

Syntaxerweiterungen

Ereigniszähler

- gesüßt

exp S

- ungesüßt

```
signal L in
    repeat exp times
      await S
    end
  ||
    ... await L; ...
end
```

Syntaxerweiterungen

Beispiel für Ereigniszähler

- gesüßt

```
loop
 await 1000 MILLISECOND
 emit SECOND
end
```

- ungesüßt

```
loop
  signal L in
      repeat 1000 times
        await MILLISECOND
      end
    ||
      await L;
      emit SECOND
  end
end
```

Syntaxerweiterungen

Immediate-Watchdog

- gesüßt

```
do
   stat
watching immediate S
```

- ungesüßt

```
present S else
   do
      stat
   watching S
end
```

Ausnahmebehandlung

- Ergänzung zu Traps

```
trap ALARM (combine integer with +),
     ZERO_DIVIDE, TERMINATE in
  stat
handle ALARM do stat1
handle ZERO_DIVIDE do stat2
end
```

- Lesen von Meldungen mit Operator `??`

```
??ALARM
```


multiples Await
Erweiterung
handle
Traps
Abwarten
Modularität
kopfgesteuerte Schleife
repeat
Ereigniszähler
Plain-Esterel
Ausnahmebehandlung
copymodule
eventhandler
timeout
syntaktischer Zucker
Wiederverwendung
await
immediate
Schnittstelle

Vorankommen

Formale Semantiken

- Entwicklung strikt nach formaler Semantik

- Notwendig wegen
  - **Komplexität** rekativer Syteme
  - Anforderungen an **Korrektheit**

- 2 mathematische Semantiken (*BERRY, GONTHIER*)
  - formale **Verhaltenssemantik**
  - **Ausführungssemantik**

Formale Verhaltenssemantik

- mathematische Definition der intuitiven Semantik
- basiert auf PLOTKIN's Kalkül
- stärker auf Implementierung ausgerichtet
- Fokus auf Ein- und Ausgabeverhalten
- schrittweise Programmableitung
- nicht ausführbar

Formale Verhaltenssemantik

- Ereignisse
- Historien
- Modulableitung

Formale Verhaltenssemantik

- Ereignis $E = S_1(v_1) \cdot S_2(v_2) \cdot \ldots \cdot S_n(v_n), \qquad n \geq 0$

  Folge emittierter Signale

- Signal abfragen: $S \in E, S(v) \in E$ oder $E(S) = v$
- leeres Ereignis: $E = \epsilon$

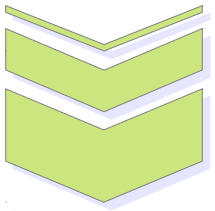
- vollständiges Ereignis $\hat{E}$

  Speichert alle möglichen Signale

  Entweder $S^+(v) \in \hat{E}$ oder $S^-(v) \in \hat{E}$

Formale Verhaltenssemantik

- Kombinationsfunktion $c(v_x, v_y)$



formaler $x \star_s y$

- synchrone Produkt $E = E_1 \star E_2$

$$S(v_1) \in E_1 \wedge S \notin E_2 \Rightarrow S(v_1) \in E$$
$$S(v_2) \in E_2 \wedge S \notin E_1 \Rightarrow S(v_2) \in E$$

$$S(v_1) \in E_1 \wedge S \in E_2 \Rightarrow S(v_1 \star_s v_2) \in E$$
$$S(v_1) \notin E_1 \wedge S \notin E_2 \Rightarrow S \notin E$$

Formale Verhaltenssemantik

- Historie $H = E_0, E_1, \ldots, E_i, \ldots$
- Vollständige Historie $\hat{H} = \hat{E}_0, \hat{E}_1, \ldots, \hat{E}_n$

$$S^- \in \hat{E}_0 \quad \Rightarrow \quad \hat{E}_0(S) = \perp$$

$$S^-(v) \in \hat{E}_i \quad \Rightarrow \quad \hat{E}_{i-1}(S) = v$$

Formale Verhaltenssemantik

Beispiel

- Signalmenge $\{S1, S2\}$
- Historie

	E_0	E_1	E_2	E_3
H	$S1(0)$	$S2(1)$	ϵ	$S1(2) \cdot S2(2)$

- vollständige Historie

	$\hat{E}_0$	$\hat{E}_1$	$\hat{E}_2$	$\hat{E}_3$
$\hat{H}$	$S1^+(0) \cdot S2^-(\perp)$	$S1^-(0) \cdot S2^+(1)$	$S1^-(0) \cdot S2^-(1)$	$S1^+(2) \cdot S2^+(2)$

Formale Verhaltenssemantik

Modulableitung

- Eingabehistorie I

  Ausgabehistorie O

  Program P

- schrittweise Überführung

$$P = P_0 \xrightarrow[\hat{I}_0]{O_0} P_1 \xrightarrow[\hat{I}_1]{O_1} \ldots \xrightarrow[\hat{I}_{i-1}]{O_{i-1}} P_i \xrightarrow[\hat{I}_i]{O_i} \ldots$$

Formale Verhaltenssemantik

Beispiel

- P_0

```
module COUNTDOWN :
  input DEC;
  output GO;
  await 3 DEC do emit GO; halt; end;
end
```

- Signal DEC tritt ein $\rightarrow$ $\hat{I}_0 = \mathrm{DEC}^+$
- P_1

```
module COUNTDOWN :
  input DEC;
  output GO;
  await 2 DEC do emit GO; halt; end;
end
```

Formale Verhaltenssemantik

Induktionsregeln

- Allgemeine Form

$$\langle stat, \rho \rangle \xrightarrow[\hat{E}]{E', b, T} \langle stat', \rho' \rangle$$

$$b = \begin{cases} tt, & stat \text{ terminiert} \\ ff, & \text{sonst} \end{cases}$$

Formale Verhaltenssemantik

- Programmüberführung

$$P \xmapsto[\hat{E}]{E'} P' \quad \Leftrightarrow \quad \langle stat, \phi \rangle \xrightarrow[\hat{E}]{E', ff, \emptyset} \langle stat', \phi \rangle$$

Ein Programm terminiert niemals

`stat` entspricht dem Körper des Programmes `P`

- Ausdrucksauswertung

$$\langle exp, \rho \rangle \xrightarrow[\hat{E}]{} \langle v \rangle$$

Formale Verhaltenssemantik

Induktionsregeln

- `nothing` Axiom

$$\langle \text{nothing}, \rho \rangle \xrightarrow[\hat{E}]{\epsilon, tt, \emptyset} \langle \text{nothing}, \rho \rangle$$

- `halt` Axiom

$$\langle \text{halt}, \rho \rangle \xrightarrow[\hat{E}]{\epsilon, ff, \emptyset} \langle \text{halt}, \rho \rangle$$

- `emit` Regel

$$\frac{\langle exp, \rho \rangle \underset{\hat{E}}{\longrightarrow} v}{\langle \text{emit S}(exp), \rho \rangle \xrightarrow[\hat{E}]{S(v), tt, \emptyset} \langle \text{nothing}, \rho \rangle}$$

Formale Verhaltenssemantik

Programmkorrektheit

- Ein Programm ist **logisch reaktiv**, wenn es mindestens eine Ausgabe für eine Eingabe liefert.
- Ein Programm ist **logisch deterministisch**, wenn es höchstens eine Ausgabe für eine Eingabe liefert.
- Ein Programm ist **logisch korrekt**, wenn es genau eine Ausgabe für eine Eingabe liefert

Formale Verhaltenssemantik

Determinismus

- Logische Kohärenzregel:

  Signal S ist vorhanden ⇔ Es gibt ein emit S

- Betrachten

```
signal S in
  present S then emit S end
end
```

Formale Verhaltenssemantik

- Annahme: S ist vorhanden

$$\frac{S^+ \in \hat{E} \qquad \langle \texttt{emit S}, \rho \rangle \xrightarrow[S^+]{E_1', b_1, T_1} \langle stat_1', \rho_1' \rangle}{\langle \texttt{present S then emit S else nothing}, \rho \rangle \xrightarrow[S^+]{E_1', b_1, T_1} \langle stat_1', \rho_1' \rangle}$$

- then-Zweig

$$\frac{\langle triv, \rho \rangle \underset{\hat{E}}{\longrightarrow} triv}{\langle \texttt{emit S(triv)}, \rho \rangle \xrightarrow[\hat{E}]{S(triv), tt, \emptyset} \langle \texttt{nothing}, \rho \rangle}$$

- Überführt

$$\frac{S^+ \in \hat{E} \qquad \langle \texttt{emit S}, \rho \rangle \xrightarrow[S^+]{S(triv), tt, \emptyset} \langle \texttt{nothing}, \rho \rangle}{\langle \texttt{present S then emit S else nothing}, \rho \rangle \xrightarrow[S^+]{S(triv), tt, \emptyset} \langle \texttt{nothing}, \rho \rangle}$$

Formale Verhaltenssemantik

- Annahme: S ist nicht vorhanden

$$\frac{S^- \in \hat{E} \qquad \langle \texttt{nothing}, \rho \rangle \xrightarrow[S^+]{E_2', b_2, T_2} \langle stat_2', \rho_2' \rangle}{\langle \texttt{present S then emit S else nothing}, \rho \rangle \xrightarrow[S^+]{E_2', b_2, T_2} \langle stat_2', \rho_2' \rangle}$$

- then-Zweig

$$\frac{\langle triv, \rho \rangle \underset{\hat{E}}{\longrightarrow} triv}{\langle \texttt{emit S(triv)}, \rho \rangle \xrightarrow[\hat{E}]{S(triv), tt, \emptyset} \langle \texttt{nothing}, \rho \rangle}$$

- Überführt

$$\frac{S^+ \in \hat{E} \qquad \langle \texttt{emit S}, \rho \rangle \xrightarrow[S^+]{S(triv), tt, \emptyset} \langle \texttt{nothing}, \rho \rangle}{\langle \texttt{present S then emit S else nothing}, \rho \rangle \xrightarrow[S^+]{S(triv), tt, \emptyset} \langle \texttt{nothing}, \rho \rangle}$$

Formale Verhaltenssemantik

- In beiden Fällen wird abgeleitet nach

```
signal S in
  nothing
end
```

- Schlussfolgerung: **Mehrfache Semantik**

- weiteres Beispiel

```
signal S in
  present S else emit S end
end
```

- Offensichtlich **ohne Semantik**

Ausführungssemantik

- garantiert Korrektheit einzelner Reaktionen
- Reaktion: Folge von atomaren Aktionen
- Ein Programm unter einer Eingabe ist korrekt, wenn es eine Ausführung gibt, in der es anhält.
- Reaktionen sind deterministisch auch dann, wenn das Programm selbst nichtdeterministisch.

Ausführungssemantik

Signale

- Implementierung als zugriffsgesteuerte geteilte Variablen
- Zugriffssteuerung über Statusinformation $\{\perp, \dagger, +, -\}$
- Signalspeicher θ speichert das Statusflag.
- Allgemeine Regel für Aktionen

$$\langle stat, \rho, \theta \rangle \rightarrow \langle stat', \rho', \theta' \rangle$$


Ereignisse
Korrektheit
logisch deterministisch
Modulableitung
Produkt
logisch korrekt
vollständige Ereignisse
Ausführungssemantik
schrittweise
Komplexität
controlled shared variable
logische Kohärenzregel
Verhaltenssemantik
Programmüberführung
formale Semantiken
synchrone
mehrfache Semantik
ohne Semantik
Induktionsregeln
Aktionen
Historien
logisch reaktiv
Statusinformationen
Überführung
vollständige Historien

Vorankommen

Armbanduhr

```
1  module WATCH:
2     type              TIME;
3     constant          INITIAL_TIME : TIME;
4     constant          ONE_SECOND, ONE_MINUTE, ONE_HOUR : TIME;
5
6     procedure         INCREMENT (TIME) (TIME),
7                       RESET_SECONDS (TIME) ();
8
9     input             SECOND, SET_HOUR, SET_MINUTE;
10    output            TIME: TIME;
11    relation          SECOND # SET_HOUR # SET_MINUTE;
12
```

Armbanduhr

```
12
13    var TIME := INITIAL_TIME : TIME in
14      emit TIME (TIME);
15      loop
16        await
17          case SECOND do
18            call INCREMENT (TIME) (ONE_SECOND)
19          case SET_MINUTE do
20            call RESET_SECONDS (TIME) ();
21            call INCREMENT (TIME) (ONE_MINUTE)
22          case SET_HOUR do
23            call INCREMENT (TIME) (ONE_HOUR)
24        end; % await
25        emit TIME (TIME)
26      end % loop
27    end % var
28  end. % module
```

Lesetipps und weiterführende Beispiele

- Ocjava: Java Code-Generierer für ESTEREL-Programme
  http://www-sop.inria.fr/meije/esterel/ocjava.html
- LEGO® MindStorm-Roboter in ESTEREL programmieren
  http://www.emn.fr/z-info/lego/
- Sehr lesenswert: Datenbank-APIs für ESTEREL
  http://drops.dagstuhl.de/opus/volltexte/2005/161/pdf/04491.WhiteDavid.Paper.161.pdf
- Weitere Beispiele: Digitaluhr, Kommunikationsprotokoll, Reflexspiel, ...
  http://www-sop.inria.fr/esterel.org/files/Html/Downloads/ProgEx/ProgExamples.htm

Fazit

- Das Einsatzgebiet von ESTEREL ist sehr spezifisch
- Die Verbreitung von ESTEREL ist (und bleibt wahrscheinlich) daher eher marginal
- Die wenigen gefundenen Quellen waren zumeist sehr theorielastig – es waren nur wenige Beispiele zu finden.
- Ein sehr interessantes Thema!

Literatur

- Berry, G.; Gonthier, G.; The **ESTEREL** synchronous programming language: design, semantics, implementation; Ecole des Mines, Valbonne, France; INRIA, Sophia-Antipolis, France; 1991; Kapitel 1 bis 8
- Berry, G.; The Esterel v5 Language Primer; Technical report; 2000
- Berry, G.; Cosserat, L.; The ESTEREL Synchronous Programming Language and its Mathematical Semantics. INRIA, Sophia-Antipolis, France; 1984
- Potop-Butucaru, D.; de Simone, R.; Talpin, J.-P.: The Synchronous Hypothesis and Synchronous Languages; IRISA, Rennes, France; INRIA, Sophia-Antipolis, France; 2004
- Berry, G. et al: The Esterel v5_21 System Manual; Ecole des Mines, Valbonne, France; INRIA, Sophia-Antipolis, France; 1999
- Dayaratne, M. W. S.; Roop, P. S.; Salcic, Z.: Direct Execution of Esterel Using Reactive Microprocessors; Dept. of Computer Engineering, University of Auckland, New Zealand; 2005. In: Electronic Notes in Theoretical Computer Science
- Palshikar, G. K.; An introduction to Esterel. Embedded Systems Programming; (November) 2001

Literatur

C. André. Representation and Analysis of Reactive Behaviors: A Synchronous Approach. Technical report, Laboratoire Informatique, Signaux, Universite de Nice-Sophia Antipolis, 1996.

J. G. P. Barnes. The standardization of RTL/2. *Software: Practice and Experience*, 10:707–719, 1980.

G. Berry. The Constructive Semantics of Pure Esterel Draft Version 3. 1999.

G. Berry. The Esterel v5 Language Primer. Technical report, 2000.

G. Berry and L. Cosserat. *The ESTEREL Synchronous Programming Language and its Mathematical Semantics*. Institut national de recherche en informatique et en automatique, 1984.

G. Berry and Esterel Team. The Esterel v5 21 System Manual. 1999.

G. Berry and G. Gonthier. The ESTEREL synchronous programming language : design , semantics , implementation. *Science of Computer Programming*, 19:87–152, 1992.

F. Boussinot and R. de Simone. The ESTEREL language. *Proceedings of the IEEE*, 79(9):1293–1304, 1991.

Literatur

M. W. S. Dayaratne, P. S. Roop, and Z. Salcic. Direct Execution of Esterel Using Reactive Microprocessors. pages 1–22, 2005.

S. A. Edwards, C. Soviani, and J. Zeng. CEC: The Columbia Esterel Compiler, 2006.

V. K. Garg. Modeling of Distributed Systems by Concurrent Regular Expressions.

T. Gautier, P. le Guernic, and L. Besnard. Signal: A declarative language for synchronous programming of real-time systems. Technical report, IRISA / INRIA, Campus de Beaulieu„ 1987.

U. Goltz. Prozeßalgebren. Technical Report November, Technische Universität Braunschweig, Institut für Programmierung und Reaktive Systeme, 2007.

U. Goltz. Vorlesung Reaktive Systeme II. Technical report, Technische Universität Braunschweig, Institut für Programmierung und Reaktive Systeme, 2008.

U. Goltz, D. Maciuszek, and W. Struckmann. Vorlesungsskript: Reaktive Systeme. Technical report, Technische Universität Carolo-Wilhelmina zu Braunschweig, 2002.

N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. Technical Report 9, 1991.

Literatur

D. Harel and A. Pnueli. On the development of reactive systems. In *Logics and models of concurrent systems*, pages 477 – 498. Springer-Verlag New York, 1989.

C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (2nd Edition)*, volume 32. Addison Wesley, 2000.

M. Jantzen, M. Kudlek, and G. Zetzsche. Concurrent Finite Automata. 2009.

F. Maraninchi. Argos: an automaton-based synchronous language. *Computer Languages*, 27(1-3):61–92, 2001.

D. May. CSP, occam and Transputers. In *Communicating Sequential Processes*, pages 75–84. Springer, 2005.

G. K. Palshikar. An introduction to Esterel. *Embedded Systems Programming*, (November), 2001.

Literatur

G. D. Plotkin. A Structural Approach to Operational Semantics. Technical report, University of Aarhus, Denmark, 1981.

D. Potop-Butucaru, R. de Simone, and J.-P. Talpin. The Synchronous Hypothesis and Synchronous Languages. pages 1–21, 2004.

S. Sergio, S. Terrasa, V. Lorente, and A. Crespo. Implementing Reactive Systems with UML State Machines and Ada 2005. *Lecture Notes in Computer Science*, pages 149–163, 2009.

C. Siemers. *Handbuch Embedded Systems Engineering*. TU Clausthal, FH Nordhausen, 2011.

E. Technologies. The Esterel v7 Reference Manual Version v7 30. Technical Report November, Esterel Technologies, 2005.

Bildquellen

- Folie 1:
  http://www.fordesigner.com/maps/15164-0.htm
- Agenda-Folien http://www.flickr.com/photos/stevebkennedy/6790930946/sizes/o/in/pool-96326254@N00/, stevebkennedy, modified
- Folie 4 (Esterel-Logo):
  http://www-sop.inria.fr/esterel.org/filesv5_92/home.htm
- Folie 5 (Logo):
  http://www.esterel-technologies.com/
- Folie 7: Flickr,
  http://www.flickr.com/photos/solapenna/8492708700/,
  Fabio Penna, retuschiert
- Folie 9: G. Morin, “Flugzeugsicherheit fürs Kraftfahrzeug,” pp. 30–33, 2006.
- Wortwolken mit wordle.net erstellt