

# Kapitel 9

# " Protocol Engineering "

9.1	Protokollspezifikation	5
9.2	Protokollverifikation	23
9.3	Protokolltest und -analyse	25

# 9. "PROTOCOL ENGINEERING"

## " Protocol Engineering " :

Technisch-wissenschaftliche Beschäftigung mit (Kommunikations-) Protokollen.

nota bene : Betonung des ingenieurmäßigen Vorgehens

Typische **Phasen des System Life Cycles**, z.B. :



→ ist auch für Protokolle gültig !

Entsprechende Begriffsdefinitionen in Anlehnung an Phasen bei allgemeiner Software- (und bedingt auch Hardware-) Entwicklung

## Def. **Protokollspezifikation** :

Formale Beschreibung von Protokollen, insbesondere die wesentlichen Aspekte einer Protokolldefinition wie

- *Syntax* und *Semantik der ausgetauschten PDUs* (Protokolldateneinheiten) sowie
- *Timing*, d.h. protokollkonforme zeitliche Abläufe während der Kommunikation zwischen den Partnern abdeckend.



## Def. **Protokollverifikation** :

Formaler Beweis der Konformität einer existierenden Protokollimplementierung mit der entsprechenden, zugehörigen Protokollspezifikation; oder – alternative Sicht – Nachweis der Konformität von gegebener Protokollspezifikation zu zugehöriger Dienstspezifikation.



**Def. Protokolltest :**

Verfahren und Maßnahmen zur Überprüfung von Eigenschaften einer Protokollimplementierung, insbesondere um den Grad der Übereinstimmung dieser Implementierung mit der zugehörigen Protokollspezifikation beurteilen zu können.

*Nota bene :*

Im Gegensatz zur Verifikation versucht die Testphase, Fehler in der Implementierung (z.B. nicht-spezifikationskonformes Verhalten) zu entdecken; kein vollständiger Nachweis der Fehlerfreiheit bei Tests !

**Def. Protokollanalyse :**

Einsatz geeigneter Analysetechniken (insbesondere Messinstrumente, Modelle) zur Bewertung der Leistungsfähigkeit oder Zuverlässigkeit von Protokollen (auf Algorithmen- oder auf Implementierungsebene).



# 9.1 Protokollspezifikation

**Anforderungen** an Spezifikationstechniken und -methoden,  
u.a. :

- Präzision und hinreichend hoher Formalisierungsgrad
- Unterstützung der Protokollimplementation  
(ideal : weitgehend automatische Umsetzung in Implementation)
- Unterstützung einer späteren Protokollverifikation
- hinreichend gute Verständlichkeit bei Interpretation der Spezifikation  
durch Menschen (z.B. Entwerfer und Implementierer des  
Protokolls)

etc.

## **Typische Probleme** von Spezifikationstechniken :

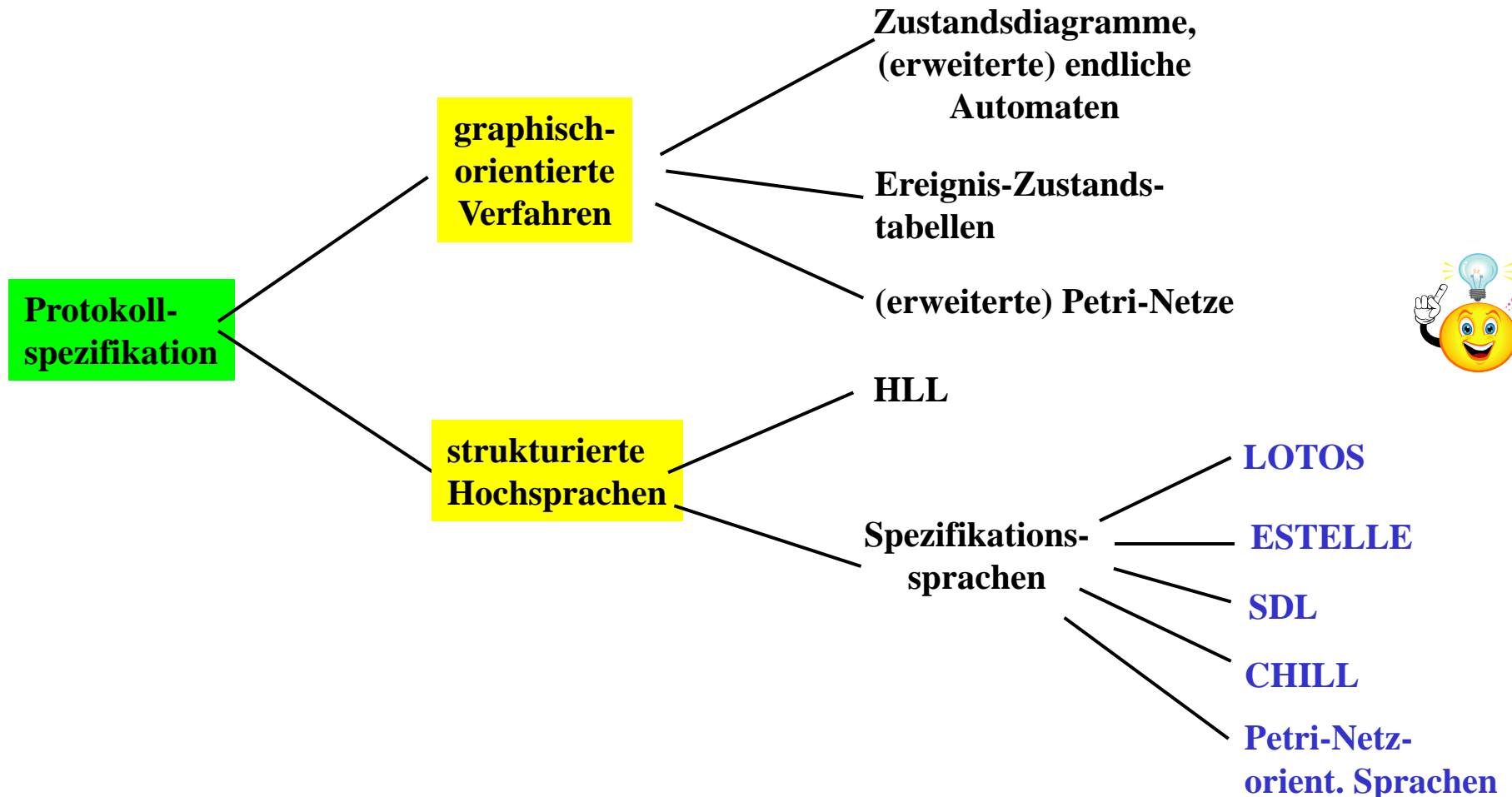
- "Trade-off" zwischen hohem Formalisierungsgrad und einfacher Verständlichkeit für Menschen
- bei Umsetzung in Implementierung → starke Betriebssystemabhängigkeiten
- signifikanter Anteil der Protokollspezifikation (i.a. >> 50% von Spezifikation) für Behandlung von Ausnahmesituationen/Fehlerfällen u.ä.

## **Angestrebtes Resultat** einer Protokollspezifikation :

präzise, vollständige, widerspruchsfreie formale Beschreibung des entsprechenden Protokolls

# Spezifikationstechniken für Protokolle

## ➤ Grobklassifikation von Spezifikationstechniken für Protokolle



## ➤ Kurzbeschreibung einiger Spezifikationstechniken

- **Zustandsdiagramme** → vgl. ausführliches Beispiel s.u.
- **Ereignis - Zustandstabellen**  
→ gleiche Information wie Zustandsdiagramme, aber Zustände und Ereignisse (die evtl. Übergänge implizieren) in Tabellenform angegeben

- **LOTOS**

→ algebraische Prozess-Spezifikationssprache  
(d.h. Spezifikation möglicher Ereignisreihenfolgen) [ISO-Standard, 1987]

Bsp.: *DataIn; Send\_Data; ACKin*

für Prozess, der abzusendende Daten (DataIn) erhält, diese sendet (Send\_Data)  
und dann auf entsprechende Quittung wartet (ACKin)



- **CHILL** (**CCITT High Level Language**) [CCITT, ~ 1979]  
→ Sprache mit ähnlichem Aufbau wie Ada, Syntax  $\approx$  Pascal, Sprachkonstrukte u.a. zur Def. von Prozessen, Beschr. von Prozesskommunikation und Ausnahmebehandlung
- **ESTELLE** → Spezifikationssprache auf Basis erweiterter endl. Automaten [ISO-Standard, 1987]: Moduln (erweiterte endl. Automaten) kommuniz. asynchron über (Kommunikations-) Kanäle, Modul-Kanalschnittstelle auf "Interaktionspunkte" abgebildet
- **SDL** (**S**pecification and **D**escription **L**anguage) [CCITT, 1989]; inzwischen: SDL-2010  
→ Details, s.u.



## Zustandsdiagramme/(erweiterte) endliche Automaten als Spezifikationstechnik

Exemplarische Illustration am **Beispiel des BSC-Protokolls**

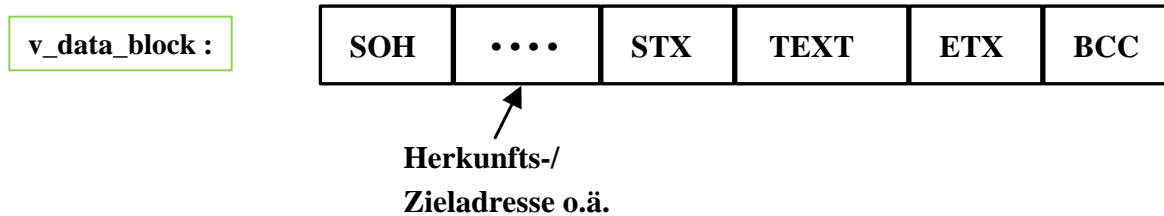
- Zustandsdiagramm für die Kommunikation zwischen einer Leitstation / „Master“ und Trabantenstationen / „Slaves“ (unter Benutzung von BSC (Binary Synchronous Communic.))
- Voraussetzung :
  - Möglichkeit der Übertragung langer Nachrichten durch Fragmentierung (Zerlegung zu langer Nachrichten in mehrere Blöcke)
  - Quittierung jedes einzelnen Datenblocks
  - Flusskontrolle mit einfachem Kredit

*Notation :*

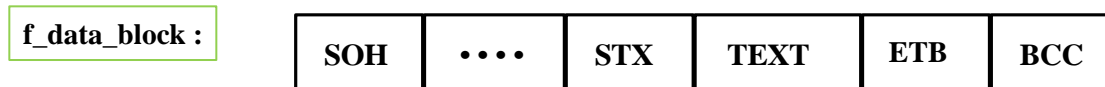
(abkürzende Schreibweise für Übertragungssteuerzeichen bzw. -blöcke)

- **data\_block (db) :**

(a) **vollständiger** Datenblock oder *letztes* Fragment bei Fragmentierung



(b) **fragmentierter** Datenblock (*nicht letztes* Fragment einer, unter Benutzung mehrerer Blöcke übertragenen, Nachricht)



- **ack :**

ACK
-----
- **nak :**

NAK
-----
- **eot:**

EOT
-----
- **poll :**

T1	ENQ
----	-----

 als Sendeaufruf (an Trabantenstation T1)
- **select :**

T1	ENQ
----	-----

 als Empfangsaufruf (an Trabantenstation T1)

- Übertragungssteuerzeichen DLE und SYN nicht berücksichtigt

Formulierung der Leit- und Trabantenstationen als **sequentielle Automaten** :

**sequentieller Automat** : 5 Tupel  $(Z, X, Y, \mu, v)$

mit  $Z$  = Menge der (inneren) **Zustände**

$X$  = Menge der **Eingabesymbole**

$Y$  = Menge der **Ausgabesymbole**

$\mu : Z \times X \rightarrow Z$  **Zustandsübergangsfunktion**

$v : Z \times X \rightarrow Y$  **Ausgabefunktion** (hier : Verwendung zur Darstellung von Aktionen)



(1) **Beschreibung der Leitstation** (vereinfacht) :

Notationen :  $\xrightarrow{\alpha/\beta}$  : Bedingung  $\alpha$  impliziert Aktion  $\beta$

EMPF( $d_1$ ) : Empfang einer Dateneinheit vom Typ  $d_1$  } *Beispiel für*  
von einer Trabantenstation } *Bedingung*

SEND( $d_2$ ) : Absenden einer Dateneinheit vom Typ  $d_2$  } *Beispiel für*  
an eine Trabantenstation } *Aktion*

Menge der Zustände der Leitstationen  $Z = \{ZL_1, ZL_2, ZL_3, ZL_4\}$

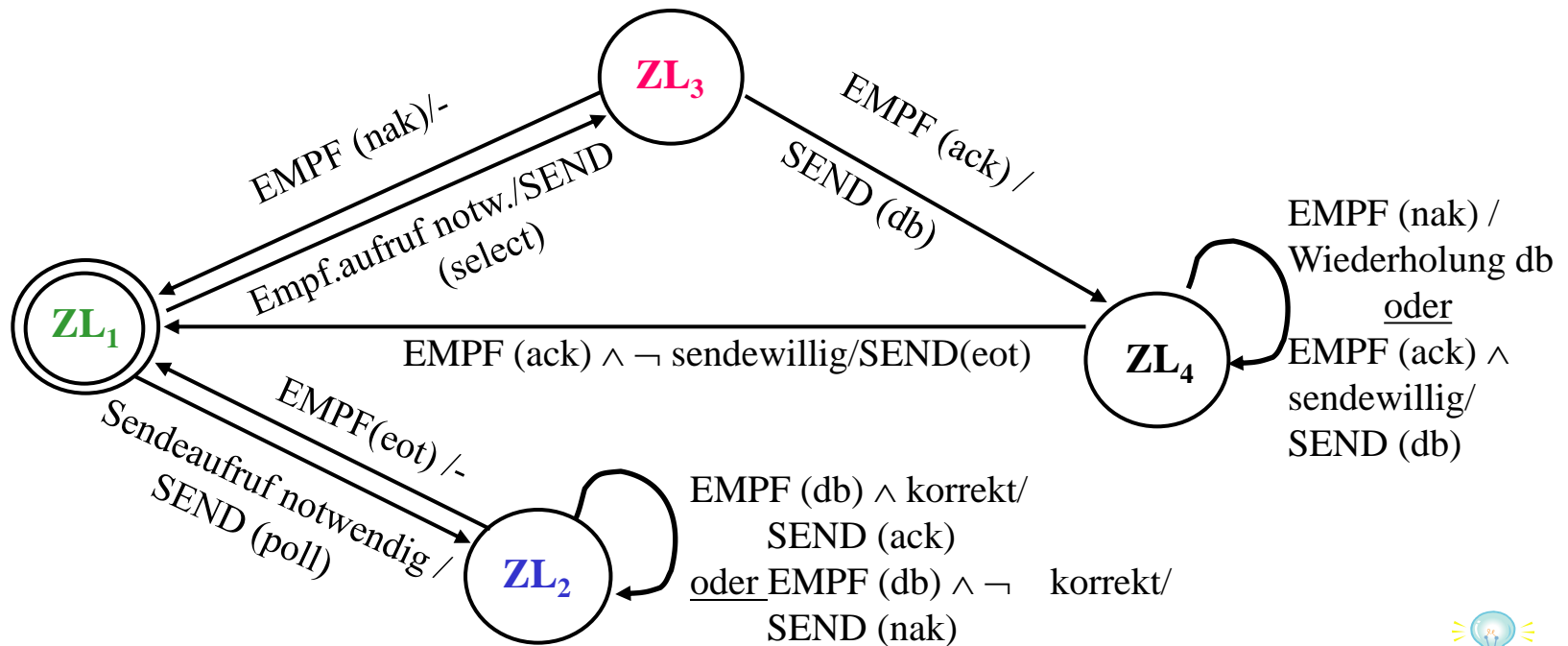
mit **ZL<sub>1</sub>** : Grundzustand (mit Möglichkeit eines Wechsels der adressierten Trabantenstation)

**ZL<sub>2</sub>** : Warten auf "data\_block" oder "eot"

**ZL<sub>3</sub>** : Warten auf Sendeerlaubnis

**ZL<sub>4</sub>** : Warten auf Quittung

## Automatengraph für Leitstation :



- ZL<sub>1</sub>** : Grundzustand
- ZL<sub>2</sub>** : Warten auf "data\_block" oder "eot"
- ZL<sub>3</sub>** : Warten auf Sendeerlaubnis
- ZL<sub>4</sub>** : Warten auf Quittung

(2) **Beschreibung einer Trabantenstation** (vereinfacht) :

*Notationen* :  $\xrightarrow{\alpha/\beta}$     Bedingung  $\alpha$  impliziert Aktion  $\beta$

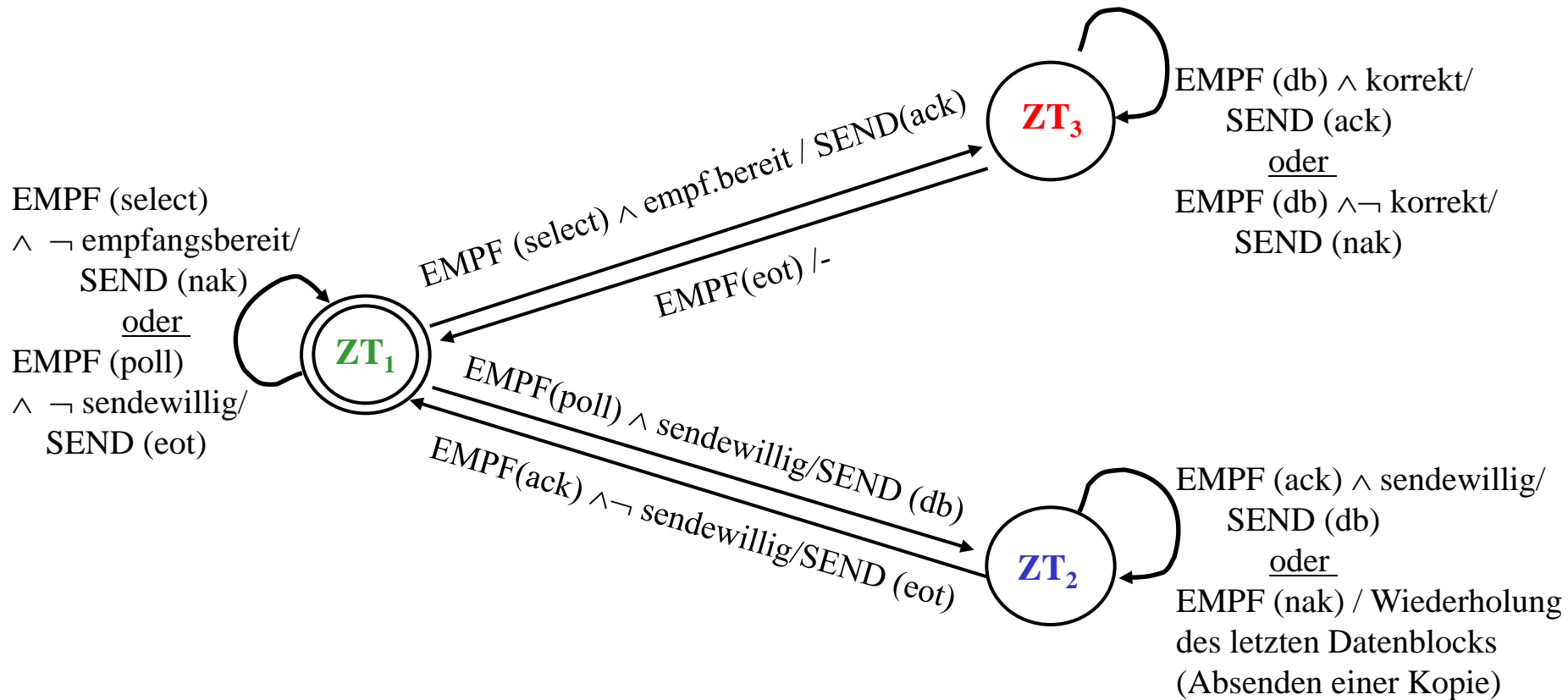
EMPF ( $d_1$ ) : Empfang einer Dateneinheit vom Typ  $d_1$   
von der Leitstation

SEND ( $d_2$ ) : Absenden einer Dateneinheit vom Typ  $d_2$   
an die Leitstation

Menge der Zustände der Trabantenstationen  $Z = \{ZT_1, ZT_2, ZT_3\}$

mit       **$ZT_1$**  : Warten auf Sende- oder Empfangsaufruf (Grundzustand)  
          **$ZT_2$**  : Warten auf Quittung  
          **$ZT_3$**  : Warten auf "data\_block" oder "eot"

## Automatengraph für eine Trabantenstation :



$ZT_1$  : Warten auf Sende- oder Empfangsaufruf

$ZT_2$  : Warten auf Quittung

$ZT_3$  : Warten auf "data\_block" oder "eot"



Vereinfachung z.B. resultierend aus Verzicht auf

- Behandlung der Zeitüberwachung
- Beschreibung der Reaktion bei Erhalt von, für den aktuellen Zustand unzulässigen Zeichen
- komplizierte Flusskontrollstrategie
- Verbindungsaufbau und -abbau
- Aussagen bzgl. Sequenz von Sende-/Empfangsaufrufen

*Nota bene:*

Weitere Erfahrungen mit automatenbasierter Protokollspezifikation unter Nutzung des bei TKRN entwickelten eLearning-Werkzeugs

## PROTOKOLLAUTOMAT

(vgl. DKR-Übungen, sofern dafür Zeit bleibt)

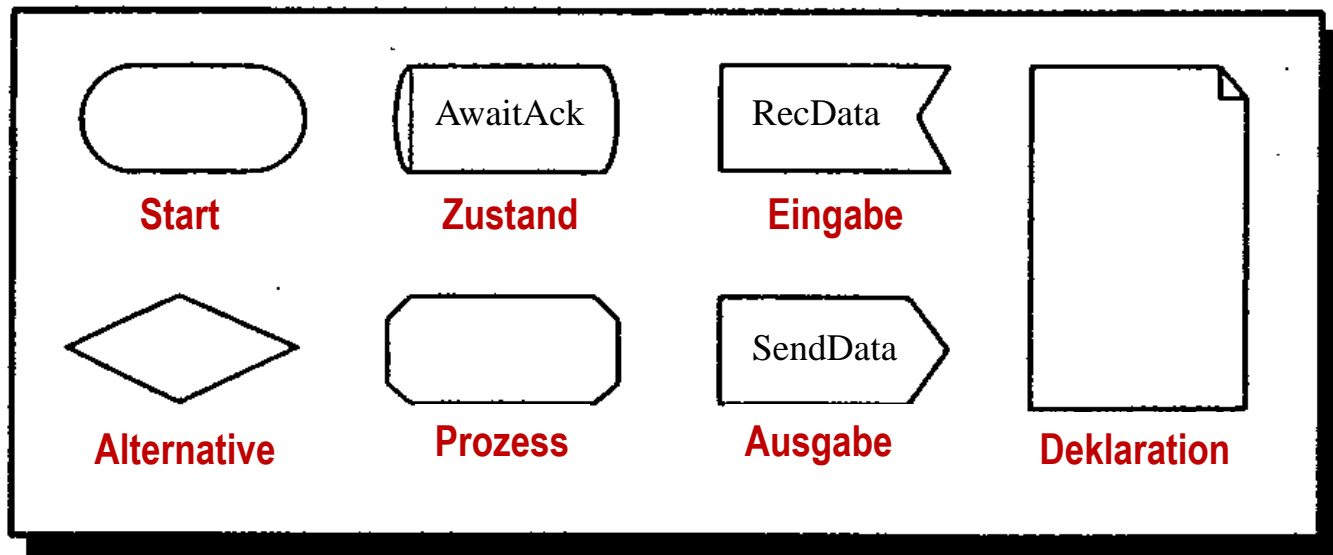
## Die Spezifikationssprache SDL

1989 : "Blue Book" des CCITT (jetzt : ITU) mit SDL als Standard - Empfehlung

Später, u.a. : SDL - 92, SDL - 96, SDL - 2000, SDL - 2010

→ SDL als praxisrelevante und verbreitete Spezifikationssprache; wie ESTELLE basierend auf erweiterten endlichen Automaten

Elementare Basiskonstrukte in SDL [vgl. Kowalk/Burke : Rechnernetze (Teubner, 1994)] :





... überdies SDL-Spezifikation basierend auf folgenden Abstraktionsebenen :

- **"System"**
- **"Block"** \*)
- **"Process"** \*) (≡ erweiterter endlicher Automat)
- **"Procedure"**



\*) : seit SDL-2000 *Blöcke* und *Prozesse* zu **"Agenten"** vereinheitlicht

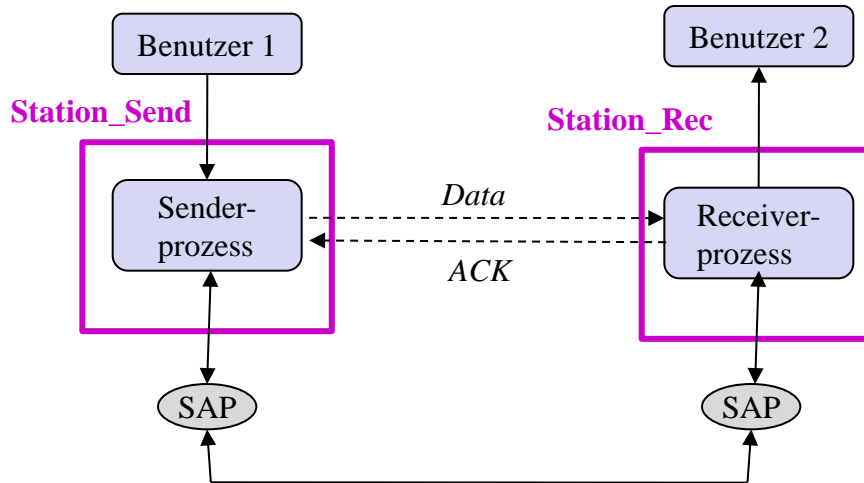
wobei :

- **System** ≡ Menge kommunizierender **Prozesse** bzw. von Subsystemen (**Blöcken**) mit ihrerseits darin enthaltenen Prozessen
- **Prozeduren** ≡ Hilfsmittel zur Strukturierung von Prozessen
- Kommunikation zwischen Prozessen über **Kanäle**
- Kommunikation mit Umwelt über **Signale**  
(z.B. den Austausch von Dienstelementen modellierend)

*Nota bene* : SDL-Spezifikation deckt ab : Verhalten einzelner Prozesse, die Kommunikation unter ihnen sowie mit Systemumgebung ("Umwelt")

# Spezifikationsbeispiel für kommunizierende Prozesse :

## Ausgangskonfiguration :

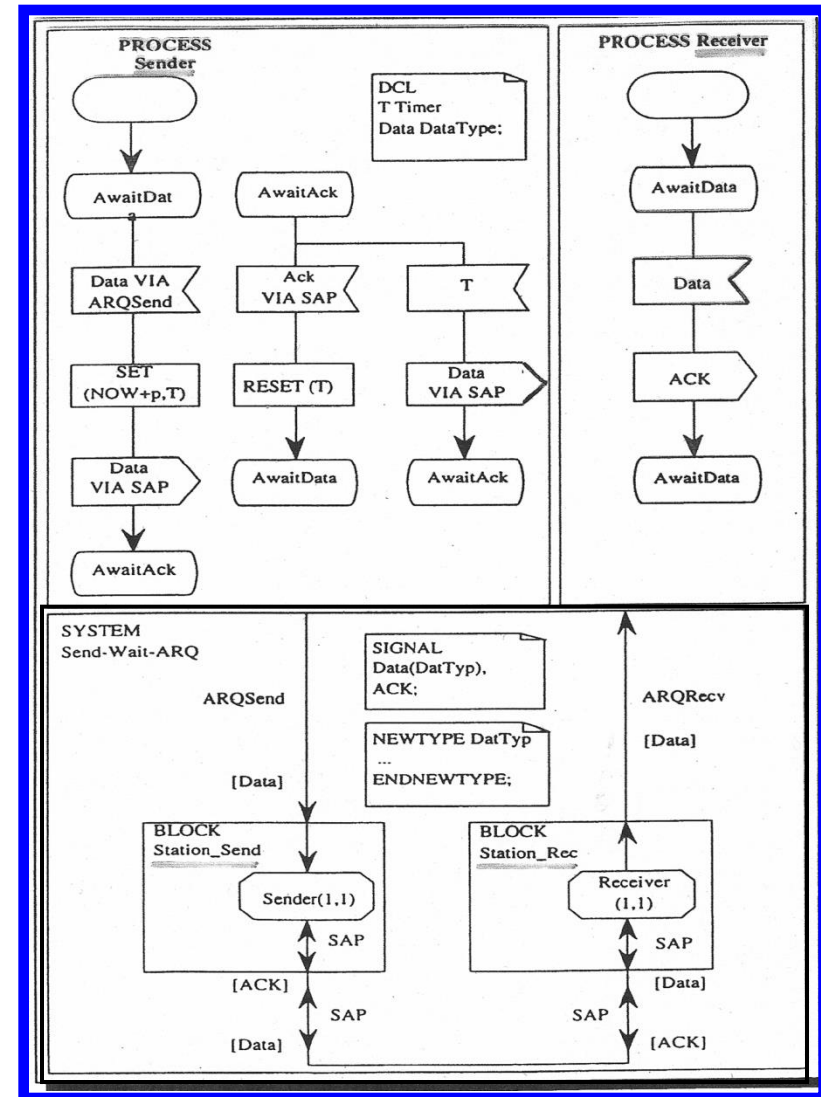


### Nota bene:

Sendende Station enthält *Senderprozess* und empfangende Station enthält *Receiver-/Empfängerprozess*, wobei der Sendeprozess Sendeaufträge aus seiner Umgebung (z.B. Benutzer des Datenübertragungs-/transportdienstes) enthält



## Zugehörige SDL-Spezifikation :


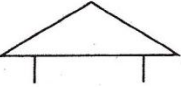
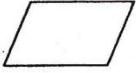








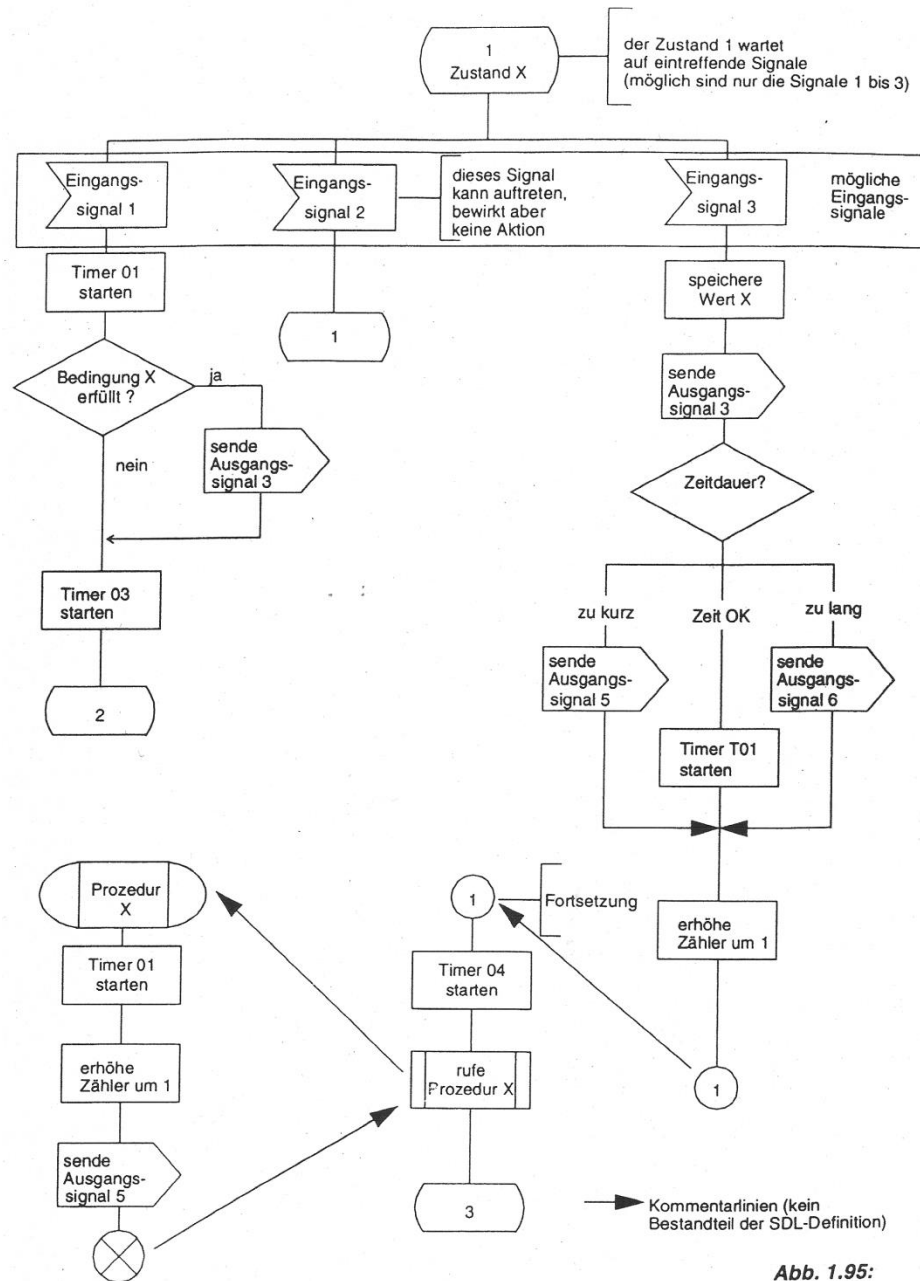
# SDL-Basiskonstrukte I (verfeinerte Sicht)



Symbol	Bezeichnung	Erlaubtes Folgesymbol	Bedeutung
	Zustand (State)	Eingangssignal	Der Zustand beschreibt eine Situation, in der ein Prozess inaktiv ist und auf ein Eingangssignal (Ereignis) wartet.[9]
	Eingangssignal (Input)	Zustand Entscheidung Aktion Ausgangssignal	Ein Eingangssignal ist jeder Anreiz, der eine Bearbeitung im System und eine damit verbundene Zustandsänderung anstößt. Der gegenwärtige Zustand wird verlassen, und die dem Eingangssymbol folgenden Aktionen werden ausgeführt.
	Ausgangssignal (Output)	Zustand Entscheidung Aktion Ausgangssignal	Ein Ausgangssignal ist eine Mitteilung des Systems an die Außenwelt (z.B. Senden einer Nachricht, Textausgabe, elektrischer Impuls).
	Aktion (Task)	Zustand Entscheidung Aktion Ausgangssignal	Unter einer Aktion versteht man systeminterne Verarbeitungsvorgänge. Entscheidungen und Ausgangssignale sind nicht eingeschlossen.
	Entscheidung (Decision)	Verzweigung Zustand Entscheidung Aktion Ausgangssignal	Die Entscheidung stellt eine logische Abfrage über die einzuschlagende Abfluss-Linie innerhalb eines Zustandsübergangs dar. [9]
	Verzweigung (Divergence)	Zustand Entscheidung Aktion Ausgangssignal	Aufteilung einer Abfluss-Linie entsprechend vorausgegangener Entscheidung.[9]
	Erzeuge-Prozess (Create Request)	Zustand Entscheidung Aktion Ausgangssignal	Der ablaufende Prozess kann bei Verwendung dieses Symbols im Verlauf einer Abfluss-Linie einen weiteren Prozess erzeugen (starten). Beim Prozessstart können Parameter mitgegeben werden. Ein Prozess kann auch zu verschiedenen Zeitpunkten, mit unterschiedlichen Parametern, erzeugt werden.
	Start		Steht am Anfang von einem Prozess, der im System erzeugt (aufgerufen) werden kann. Alle folgenden Aktionen bis zum Erreichen eines Zustands werden abgearbeitet.
	Stopp	keins	Beendet einen erzeugten Prozess.
	Prozedur-Aufruf (Process Call)	Entscheidung Aktion Ausgangssignal	Ruft eine Prozedur (Unterprogramm) auf (keine Zustände).
	Prozedurstart (Process Start)	alle	Start eines Unterprogramms.

# SDL-Basiskonstrukte II (verfeinerte Sicht)

Symbol	Bezeichnung	Erlaubtes Folgesymbol	Bedeutung
	Prozedur-Stopp (Process Stop)	keins	Stopp einer Prozedur und anschließender Rücksprung zum Punkt, an dem die Prozedur aufgerufen wurde.
	Option	alle	Wird in Spezifikationen verwendet, die Auswahlmöglichkeiten bieten, wie ein bestimmter Ablauf zu implementieren ist.
	Wartebedingung (Save)	Zustand	Die Wartebedingung stellt das zeitliche Zurückstellen eines Eingangssignals dar, wenn der Prozess sich in einem Zustand befindet, in dem die Wahrnehmung dieses Eingangssignals nicht vorgesehen ist. [9]
	Ablauffluss-Linie	alle	Verbindungselement zwischen den Symbolen, über welche die Folge des Ablaufs festgelegt ist.
	Zusammenführung (Convergence)	alle	Vereinigung zweier Ablauffluss-Linien.
	Verbindungselement (abgehend) (Connector)	keins	Eine Ablauffluss-Linie kann zur übersichtlichen Unterteilung durch ein abgehendes Verbindungselement unterbrochen werden, der Ablauffluss wird beim zugeordneten ankommenden Verbindungselement fortgesetzt.
	Verbindungselement (ankommend)	alle	Fortsetzung einer zuvor unterbrochenen Ablauffluss-Linie.
	Kommentar (Comment)	–	Erklärende Zusatzinformationen zur besseren Lesbarkeit oder zum besseren Verständnis von SDL-Diagrammen.
	Kennzeichnung intern erzeugter Eingangssignale	–	Wird in ETSI-Spezifikationen zur Kennzeichnung intern erzeugter Eingangssignale verwendet.



**Abb. 1.95:**  
Beispiel eines  
SDL-Diagramms

## Zur Analyse von Spezifikationen

Angestrebte Aussagen :

➤ **Deadlockfreiheit für spezifiziertes Protokoll**

→ Verklemmungssituationen möglich ?

(bei zulässigem oder gar unzulässigem Benutzerverhalten, sofern letzteres nicht auszuschließen)

➤ **Fairnesseigenschaften**

→ Benutzer fair von Protokoll (bzw. entsprechendem Dienstbringer) behandelt, dergestalt, dass nicht Aufträge eines Benutzers auf Dauer durch andere Benutzer verdrängt werden können

➤ **Vollständigkeit**

→ insbesondere auf alle möglichen Ereignisse (auch bei Auftreten von Fehlersituationen) Reaktionen vorgesehen

➤ **Lebendigkeit**

→ Abwesenheit von partiellen Verklemmungssituationen (z.B. Protokollgraph nach partieller Verklemmung nur noch unvollständig durchlaufen, Endlos-Schleifen u.ä.)

➤ **Leistungsbewertung auf Spezifikationsebene**

→ vgl. Protokollanalyse (z.B. Ermittlung der Menge auszutauschender PDUs zur Lösung einer Kommunikationsaufgabe)

u.a.m.





# 9.2 Protokollverifikation

Zu beantwortende Frage :

Besitzt eine vorliegende Protokollimplementation sämtliche der durch die Protokollspezifikation geforderten Eigenschaften ?

→ hier besonders wichtig :

spezifizierte Eigenschaften durch Implementation nachweisbar/beweisbar erfüllt !  
(*nota bene*: **zu verifizierende Eigenschaften** häufig **Deadlockfreiheit**, **Fairness** und **Lebendigkeit** einschließend, bei Echtzeitkommunikation evtl. überdies auch **spezifizierte Leistungs-** bzw. **Dienstgüte-/QoS-Anforderungen** zu erfüllen)



Protokollverifikation vereinfacht durch

- automatische Umsetzung von Spezifikation in Implementierung  
(ABER : Betriebssystemabhängigkeit s.o.)
- Beschränkung auf sehr einfache Protokolle (dann jedoch NICHT praxisrelevant !)
- Nutzung von (höheren Programmier- oder speziellen Spezifikations-) Sprachen anstelle von graphik-orientierten Verfahren → insbesondere graphische Verfahren zum Teil direkt auf sprachliche Verfahren abbildbar

Zu Details bzgl. Spezifikation und Verifikation von parallelen und verteilten Programmen vgl. auch einschlägige Vorlesungen der TGI-Gruppe unseres Fachbereichs !

Nota bene :

- 1) Da Protokollimplementierung speziell auch ein Programm repräsentiert  
→ möglicher Einsatz von Programmbeweisern  
(zum Nachweis spezifizierter Programmeigenschaften)
- 2) In Literatur zum Teil Verwendung des Begriffs "**Programmverifikation**"  
als Ersatz für den hier verwendeten Begriff "**Protokollverifikation**" und  
dort dann :

**Protokollverifikation** als Nachweis, dass Protokollspezifikation die  
Dienstspezifikation erfüllt.



## 9.3 Protokolltest und -analyse

### Testen von Programmen

(insbesondere von implementierter Protokollsoftware):

Kontrollierte Ausführung des Programms mit ausgewählten Sequenzen von Eingabedaten.

Bei Test im allg. :

- möglicher Nachweis der Anwesenheit von Fehlern, *wohingegen*
- Nachweis der Abwesenheit von Fehlern unmöglich !

## "Debugging" :

Vorgehensweise zur Suche der genauen Fehlerursache bei Auftreten von Fehlern (während Tests)

➤ *Varianten des "Debugging":*

- **statisches Debugging** → Suche von Fehlern in Programmdokumenten (ohne Programmausführung)
- **dynamisches Debugging** → kontrollierte Programmausführung (incl. geeigneter Programminstrumentierung) zum Zwecke einer Fehlersuche

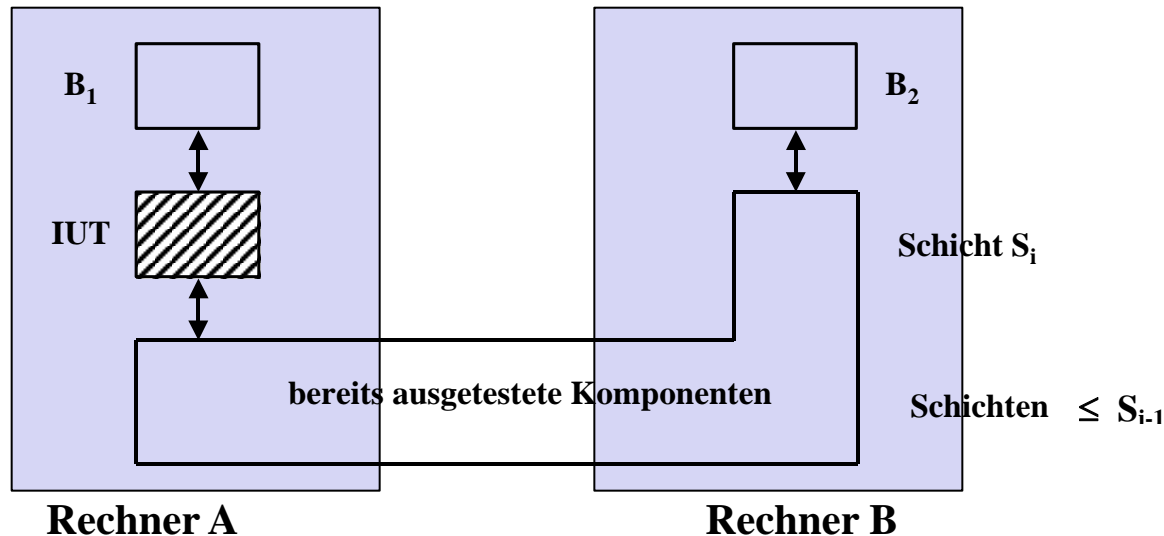


➤ *Anforderungen an Testhilfen ("Debugger") u.a.*

- Benutzerfreundlichkeit, wie z.B. intuitive Benutzbarkeit, Benutzerführung, ...
- Reproduzierbarkeit → möglichst Wiederholbarkeit von Debugging-Sitzungen (ergo: notwendige Einschränkungen von "Zufälligkeiten", indes "zeitliche Verzahnung" bei Nebenläufigkeit evtl. nur schwer reproduzierbar)
- angemessene Wiedergabe von Zustandsinformation, z.B. bzgl. interner Programzustände sowie Info über Kontrollflüsse
- Möglichkeit der Modifikation von Systemzuständen, z.B. zur gezielten Erzeugung spez. Randbedingungen zur Umsetzung der angestrebten Teststrategie etc.

## Testen von Protokollimplementationen

*Typischer Ansatz :*



**IUT** : Instance Under Test  
(zu testende Komponente)

→ in  $B_1$  und in  $B_2$  spezielle Typen und Sequenzen von Aufträgen zu generieren, um gezielt ein bestimmtes Verhalten der Instanzen auf Schicht  $S_i$  (sowie den unterliegenden Schichten) zu erzwingen ! Zusätzlich evtl. auch Verhalten der unterliegenden Schichten variieren (z.B. hinsichtlich Verlusten / Verzögerungen von Dateneinheiten)

## Sinnvolle **Teststrategien und -methoden** (im o.g. präsentierten “IUT-Ansatz“) **u.a.**

- “Loop-Back” -Test : Basisdienst sendet erhaltene Dateneinheit an ursprünglichen Sender (IUT) zurück
- Vorgabe von spezifischen Lastsituationen/Benutzerverhalten (B<sub>1</sub>-seitig) → Ziel : Durchlaufen möglichst vieler (Programm-)Pfade, z.B. in Protokollautomat
- B<sub>2</sub>-seitige Variationen
- gezielte Manipulationen an dem benutzten Basisdienst, z.B. zur Erzeugung spezieller Fehlersituationen
- gezielte Variation des Timings, insbesondere zur Entdeckung zeitabhängiger Fehlersituationen
- Erzeugung eines unzulässigen Umgebungsverhaltens (bezogen auf IUT-Umgebung) → unerwartete “Grenzsituationen”

Generell anzustreben : automatische Generierung geeigneter Testsequenzen

---

### **Protokollanalyse :**

... in der Regel bezogen auf Analyse von Leistungsfähigkeit, Zuverlässigkeit, Verlässlichkeit, QoS, o.ä.  
(seltener gemeint : Analyse von Fehlersituationen)

→ auch bei Protokollanalyse : evtl. gezielte Experimente mit implementierter Protokoll-SW unter Einsatz von Messwerkzeugen

*Nota bene :*

Zu einer detaillierten Diskussion hinsichtlich Protokoll- und Netzanalyse, vgl. Kap. 11