

**Alternating Bit Protocol (ABP):**

*Voraussetzungen:*

- Protokoll für die Übertragung von Nachrichten von einem Sender zu einem Empfänger
- Sender und Empfänger sind über einen Nachrichten-Kanal und einen Bestätigungskanal miteinander verbunden (also 4 Komponenten: Sender, Empfänger, Nachrichten-Kanal, Bestätigungskanal (acknowledgement channel))
- Annahme aus der Realität, die Kanäle sind unzuverlässig, d.h. sie können Nachrichten verlieren oder verdoppeln

*Abläufe:*

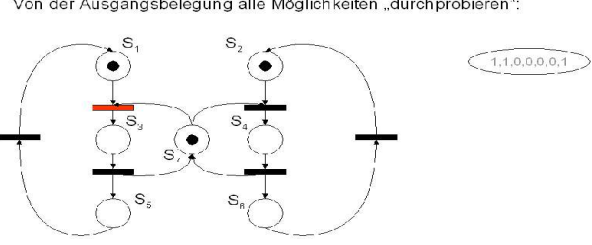
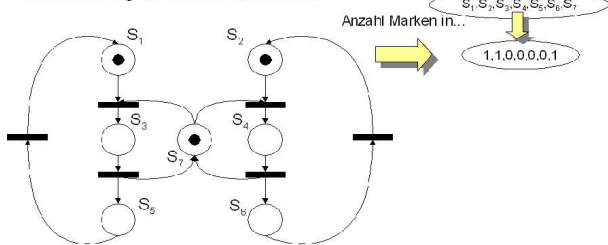
- Der Sender sendet eine Nachricht zusammen mit einem Kontrollbit
- Wenn der Empfänger eine Nachricht erhält, so sendet er das Kontrollbit dieser Nachricht als Bestätigung zurück, sofern es dem Bit entspricht, dass vom Empfänger erwartet wurde, andernfalls sendet er das alte Bit, der letzten Nachricht.
- Bekommt der Sender das richtige (d.h. das erwartete) Bestätigungsbit („ack bit“), so ändert er das Kontrollbit für die nächste Nachricht. (daher „alternating bit protocol“).
- Bekommt der Sender das falsche (also ein nicht erwartetes) Bestätigungsbit, so wiederholt der Sender die alte Nachricht samt Kontrollbit noch einmal.

*Sinn/weitere Informationen:*

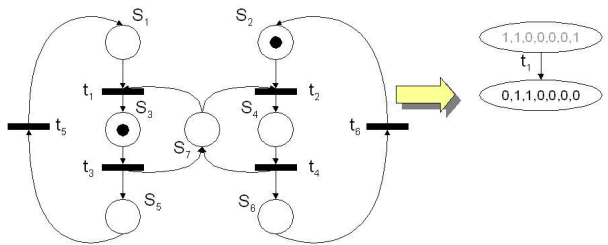
- durch das alternierende Kontrollbit können Sender und Empfänger den Verlust einer Nachricht, oder die Verdopplung einer Nachricht entdecken
- es lässt sich nur erkennen, ob die richtige Nachricht versandt wurde, also nicht ob die Nachricht auch korrekt übermittelt wurde. Um solch Korruptionen zu entdecken müssen andere Methoden angewandt werden (z.B. Paritätsbit, Quersummen)

**Erreichbarkeitsgraph:**

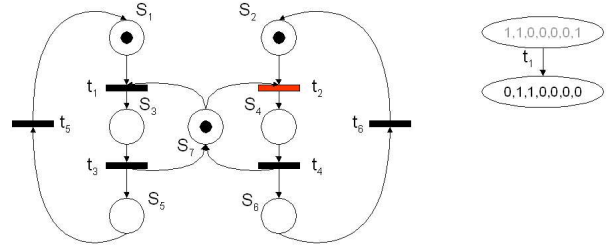
Nummerierung der Stellen erforderlich:



Von der Ausgangsbelegung alle Möglichkeiten „durchprobieren“:

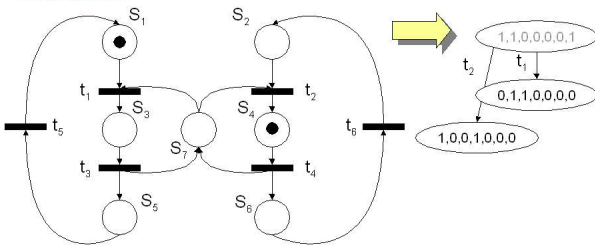


Von der Ausgangsbelegung alle Möglichkeiten „durchprobieren“:

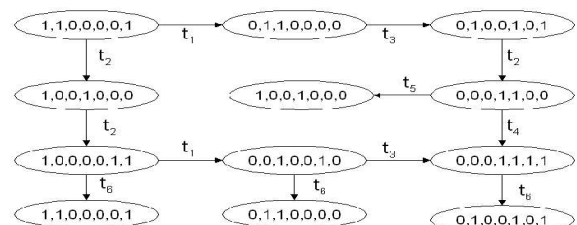


Von der Ausgangsbelegung alle Möglichkeiten „durchprobieren“:

2. Alternative:

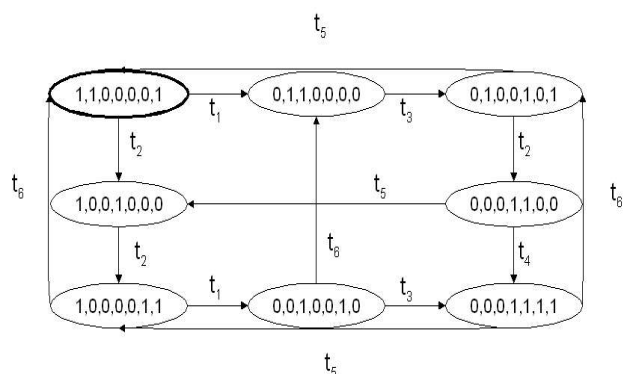
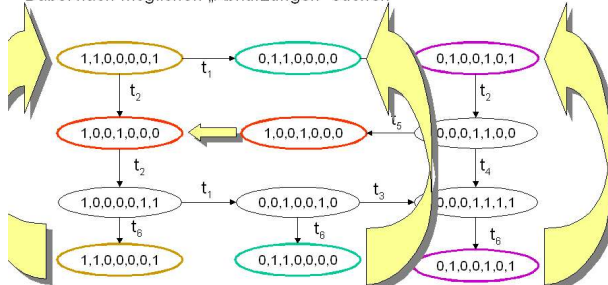


Von den Folgebelegungen auch alle Möglichkeiten „durchprobieren“:



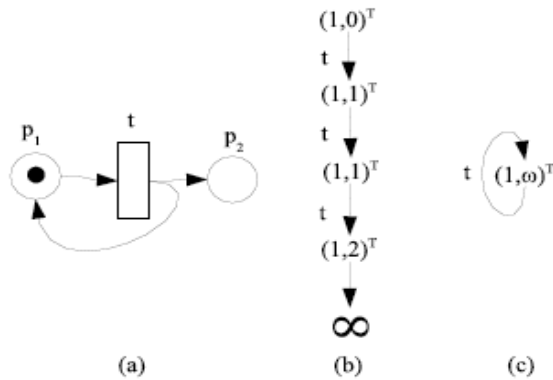
Von den Folgebelegungen auch alle Möglichkeiten „durchprobieren“:

Dabei nach möglichen „Abkürzungen“ suchen



**Überdeckungsgraph:**

- bei Rückkopplungen ratsam, da Erreichbarkeitsgraphen hierbei divergieren können, und so lassen sie unter Umständen keine Aussagen mehr über die Erreichbarkeit von Zuständen zu.



**Inzidenzmatrix:**

Eine **Inzidenzmatrix** zu einem Graph mit  $n$  Knoten und  $m$  Kanten ist eine  $n \times m$ -Matrix, bei der die Zeilen mit den Knoten und die Spalten mit den Kanten identifiziert werden. Dazu nummeriert man die Knoten von 1 bis  $n$  und die Kanten von 1 bis  $m$  durch und trägt in die Matrix die Beziehungen der Knoten zu den Kanten ein.

In den Spalten werden die Transitionen (Vierecke) abgetragen, und in den Zeilen die Plätze.

An den einzelnen Positionen in der Matrix, steht das Gewicht der Kanten, die beide verbindet.

Bei Stellen eingehende Kanten mit positivem Vorzeichen, Ausgehende Kanten mit negativem Vorzeichen. Nicht vorhandene Kanten sind 0.

**Folgemarkierung:**

Eine Folgemarkierung stellt wenn man eine aktive Transition „feuern“ lässt, das Ergebnis nach diesem Vorgang dar. Folgemarkierungen lassen sich mit Hilfe der Inzidenzmatrix berechnen.

Berechnung der Folgemarkierung  $M'$ :

$$M_0 + C * w = M'$$

$$\begin{pmatrix} 2 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix} * \begin{pmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$w$  ist hierbei ein Vektor, in dem alle Transitionen stehen die geschaltet werden, natürlich mit  $w = (t_1, t_2, \dots, t_n) \dots$  die Ziffern an den jeweiligen Positionen der Transitionen beschreibt dann wie häufig diese Transition „gefeuert“ wird.

**S-Invarianten (Stellen) und T-Invarianten (Transitionen):**

- Dienen zum Nachweis von Eigenschaften
- können aus der Netzstruktur abgeleitet werden
  - T-Invarianten sind Vektoren, die oben in die Gleichung eingesetzt werden können für  $w$ , sie sagen welche Transitionen gefeuert werden müssen, um die selbe Markierung wie die aktuell vorliegende erreichen zu können.
    - gesucht ist also ein „ $w$ “ das die Gleichung  $C * w = 0$  erfüllt (resultierend aus  $M' = M_0$ )
    - $w$  muss ganzzahlig und nichtnegativ sein
  - S-Invarianten sind Vektoren von Stellen (wie es auch  $M_0$  und  $M'$  sind), die beim Schalten von Transitionen in der Summe der Markierungen unverändert bleiben
    - mit der Transponierten von  $C$  also  $C^T$  können die S-Invarianten direkt berechnet werden, sie erfüllen wenn die Invariante  $y$  sei folgende Gleichung:  $C^T * y = 0$
    - da Markierungen stets ganzzahlig und nichtnegativ, so muss auch  $y$  ganzzahlig und nichtnegativ sein.

$$D_0 = \left( \begin{array}{ccc|ccc} -1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 1 \end{array} \right) \begin{array}{l} - \text{Lösungsschema: man schreibe die Matrix } C \text{ und die Einheitsmatrix} \\ \text{nebeneinander, ... dann versuche man durch Addition der Zeilen Nullzeilen zu erzeugen.} \\ - \text{auf der rechten Seite, mit der Einheitsmatrix, entstehen in diesen Zeilen die} \\ \text{Invarianten} \end{array}$$

$$D_1 = \left( \begin{array}{ccc|ccc} 0 & -1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array} \right) \begin{array}{l} - \text{Zeilen die nicht Null werden können, liefern keine Invariante.} \\ - \text{für die S-Invariante (auch P-Invariante) muss man die Matrix vorher transponieren} \end{array}$$

$$D_2 = \left( \begin{array}{ccc|ccc} 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{array} \right)$$

**Trap's:** Sind „Schleifen“, das meint das Stellen denen eine Markierung abgenommen wird durch feuern einer Transition diese entnommene Markierung wieder zurück bekommt. (auch über mehrere Transitionen und Stellen möglich)

**Fallen:** Ob Fallen vorliegen kann man überprüfen, indem man die Markierten Knoten nimmt, und dann die Menge der eingehenden mit der Menge der ausgehenden Kanten vergleicht. Hierbei dürfen die Transitionen der Ausgangsmenge nicht vollständig in der Transitionsmenge der eingehenden Kanten enthalten sein.

Beispiel:  $\{p_1, p_2, p_3\} * = \{t_1, t_2, t_3\} \not\subseteq * \{p_1, p_2, p_3\} = \{t_2, t_3, t_4\}$

Diese Markierung ist keine Falle, da die Mengen der Transitionen, der ausgehenden Kanten nicht in der Menge der eingehenden Kanten enthalten ist. Denn sie enthält zusätzlich die Transition t1.

**OCL (Object Constraint Language):** (Wird wohl nicht drankommen, geht zu sehr in das Fachgebiet der Programmentwicklung)

- formale Sprache ohne mathematischen Hintergrund
- wird eingesetzt in objektorientierten Modellen
- Seiteneffektfrei

Wozu?

zur Definition von Rahmenbedingungen (constraints) (Vor- & Nachbedingungen, Invarianten) in Standards

Datentypen:

Collection Typ Set ist Menge von Elementen ohne Duplikate

Collection Typ Bag ist Menge von Elementen unter Umständen mit Duplikaten

Collection Typ Sequence ist Menge von Elementen unter Umständen mit Duplikaten in geordneter (nicht sortierter) Form

**Moore-Automat:** Beim Moore Automat steht im Knoten, was bei Erreichen des Knotens ausgegeben werden soll, und an den Kanten zu welchem Knoten gegangen werden soll, wenn das entsprechende Zeichen an der Kante gelesen wird. Erst lesen dann schreiben :).

**Mealy-Automat:** Hierbei bedeutet die Kantenbeschriftung 0/1, dass bei Eingabe einer Null zusätzlich zum Wechsel des Zustands eine Eins ausgegeben wird.

**State-Chart:**

- betrachtet den Verhaltensaspekt
- beschreibt dynamisches Verhalten auf interne und externe Einflüsse und die Beziehungen zwischen den einzelnen Funktionen
- Verallgemeinern die Darstellung endlicher Automaten durch stark erweiterte Zustandsübergangsdiagramme
  - Einführung von Hierarchie und parallelen Vorgängen
  - Kontrolle des dynamischen Verhaltens, und der Informationsflüsse

**Sequence-Chart:**

Kommunikation zwischen Prozessen:

- synchron --> Pufferkapazität = 0
- beschränkt asynchron --> Pufferkapazität = n
- asynchron --> Pufferkapazität nicht vorgegeben also unbeschränkt

Nachrichtenreihenfolge:

- entweder man legt fest, Nachrichten können sich überholen oder
- man geht nach FIFO vor (First In First Out)

Semantik:

- **Die Kausale Ordnung:**  $\leq_K$  ist die reflexive transitive Hülle  $\rightarrow_K$ 
  - Die Kausale Ordnung bei Fifo und asynchroner Kommunikation ist gewährleistet wenn:
    1. Send Ereignis einer Nachricht ist vor dem Receive Ereignis
    2. Send-Ereignisse eines Prozesses werden geordnet
    3. Nachrichten in einem Puffer können sich nicht überholen.
  - effiziente Berechnung mit Warshall Algorithmus ( $O(n^3)$ )
- Notation:
  - $r_{ij}$  ist ein receive-Ereignis zwischen Prozess i und j
  - $s_{ij}$  ist ein send-Ereignis zwischen Prozess i und j
  - der Inhalt der Ereignisse (Nachrichten) ist in der Regel irrelevant, es geht um die Tatsache des Verschickens von Nachrichten überhaupt.
  - $w|_P$  bezeichnet die Projektion auf die Ereignisse von Prozess P (alle send-Ereignisse von P aus und alle Receive-Ereignisse mit Ziel P)
- **Linearisierung** ist eine totale Ordnung der Ereignisse, die die kausale Ordnung  $\leq_K$  enthält
- **wohlgeformt** die Anzahl der receive-Ereignisse ist höchstens so groß wie die der Send-Ereignisse (im Präfix)
- **Race Conditions (Konkurrenzsituation)...???**
- **Realisierung von MSC (Message Sequence Charts):**
  - kommunizierender Fifo-Automat????

**Alloy:**

Schlüsselworte:

sig = Signatur = Definition einer Klasse

option = optional = kann muss aber nicht, zur Klassifizierung von Attributen eingesetzt

set = Ansammlung = mehrere Elemente

extends = Vererbung = Erweitert

iden[blub] = dies speziell durch blub identifizierte Objekt

(A --> B) Erbt eine Klasse von einer anderen Klasse, so gehört jedes Objekt beiden Klassen an. Hierbei ist es nun Möglich, mit dieser Anweisung zu überprüfen, wenn A die Vaterklasse ist, ob B von A geerbt hat. Hierbei liefert die Aussage keinen boolean zurück, sondern das Objekt, wenn die Aussage zutrifft.

one = ein einzelner

some = einige

all = alle

no = kein

| = für die gilt

|| = oder

& = und zusätzlich gilt (dieser Befehl siebt die Menge aus)

&& = die Linke Menge plus der Rechten (dieser Befehl fügt Mengen zusammen)

p++q = Fügt zwei Relationen zusammen, wenn ein element durch p abgedeckt ist, und nicht durch q so werden die Überstehenden Elemente, zur Menge der Relation die durch q bestimmt ist hinzugefügt.

=> = impliziert

static = Statisches Element also quasi ein Objekt der Klasse

fact = Schlüsselwort für den Beginn einer „Funktion“ = die Funktionen erzeugen die Umgebung in der ner Elemente existieren können, die die Fakten erfüllen

fun = function

run = Ausführen

in = is element aus der Menge von

\* = Alle Objekte die die angegebene Eigenschaft erfüllen, das aktuelle wird nicht berücksichtigt

~ = Alle Objekte die die angegebene Eigenschaft nicht erfüllen-> kehrt die Beziehung um

^ = Alle Objekte die die angegebene Eigenschaft erfüllen, das aktuelle wird berücksichtigt

## 0. Basisklassen

Sind Signaturen,... interpretierbar ähnlich den Klassen in Java... diese Klassen können Attribute besitzen, die dann in Klammern angefügt sind.

Beispiel: sig Person {spouse: option Person, children : set Person}

Erklärung: es wird eine Klasse Person definiert, diese kann einen Spouse (Ehepartner) besitzen, muss aber nicht. außerdem kann eine Person mehrere Kinder haben.

### 1. Relationen

Definiert Spezielle Teilmengen aus der durch die Signatur vorgegeben Mengen. Diese Teilmengen erfüllen spezielle Eigenschaften.

Beispiel: parents = ~children

schwiegereltern = spouse.~children

schwiegermutter = spouse.parents & (Person->Woman)

Erklärung:

1. Eltern sind alle außer den Kindern also quasi Gegenteil der Kinder
2. Schwiegereltern sind des Ehepartner Eltern, also des Ehepartners nicht Kinder also des Ehepartners das Gegenteil der Kinder
3. Schwiegermutter ist von den Eltern des Ehepartners die Frau, wir sieben also alle aus die nicht dazu passen, und über bleiben die Passenden Objekte

### 2. Aussagen

Sind Sätze die eine logische Kombination der Relationen darstellen, oder anders gesagt komponierte Relationen, mit Mengenbeziehungen.

Beispiel: Manche Leute haben Schwiegereltern.

some p: person | some p.spouse.parents

es gibt einige p von der Klasse Person für die gilt einige von diesen p's haben

Ehegatten mit Eltern

### 3. Funktionen

sind Fakten, Fakten ergeben sich durch Konkatenationen von Aussagen zum Beispiel, und stellen Quasi Funktionen dar, die ausfiltern was nicht passt, und eine Umgebung schaffen.

Beispiel:

```
fact Facts {  
  no p:Person | p in p.^children  
  no p:Person | p in p.spouse  
  all m:Man | (m.spouse in Woman) || no m.spouse  
  all w:Woman | (w.spouse in Man) || no w.spouse }
```

Es gibt keine Person p die gleichzeitig in ihren Kindern enthalten ist, oder in den Kindern der Kinder, oder den Kindern der Kinder der Kinder usw. also alle Nachfahren.

Es gibt keine Person p die gleichzeitig sein Ehegatte ist. Also niemand ist mit sich selbst verheiratet

Für Alle Männer m gilt, dass der Ehegatte aus der Menge der Frauen stammt oder der Mann besitzt keinen Ehepartner

Siehe Männer.

#### 4. Module

Sind Verbundstrukturen, aus Punkten 0-3 , ähnlich Ada-Packages oder so.