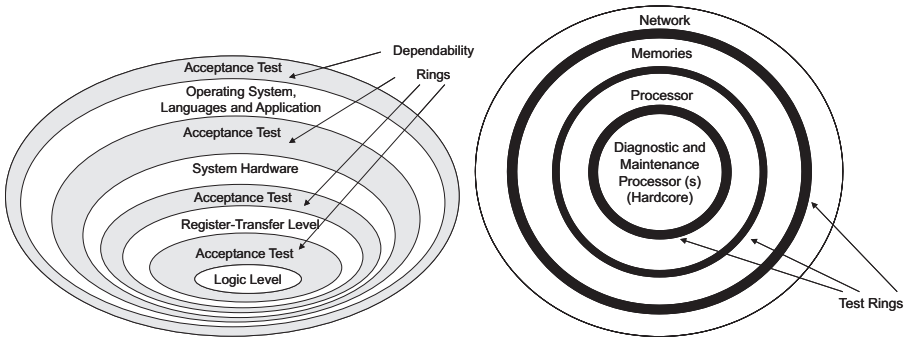
Verlässliche Systeme

7. Kapitel
Diagnose

Prof. Matthias Werner
Professur Betriebssysteme

7.1 Einführung

- Diagnose ist ein Standardansatz beim Entwurf verlässlicher Systeme
- Einsatz:
  - Fehlerdiagnose zur Fehlerbehebung
  - Fehlerdiagnose zum Fehlerabschottung (Verhinderung von Fehlerpropagierung)
- Abschottung (*Containment*) kann in horizontalen oder vertikalen Schichten erfolgen:



Tests

- Diagnose hat mehrere Aspekte
  - Erkennung, dass ein Fehler vorliegt ➡ **Fehlererkennung (fault detection)**
  - Lokalisierung des Fehlers
- Fehlererkennung wird mit Hilfe von **Tests**/**Checks** durchgeführt
- Bewertungskriterium für einen Test ist die **Abdeckung** ➡ siehe 3.12

Achtung

Im Fall dass ein Test einen Fehler anzeigt, muss dies nicht bedeuten, dass der Testkandidat fehlerhaft ist.
Es kann auch der Tester sein!

- Mehr im Abschnitt zu Systemdiagnose

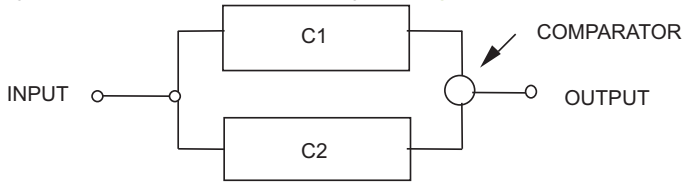
7.2 Standardtechniken der Diagnose

Test/Checks

- **Standard checks**
  - Replikationsschecks
  - Timingchecks
  - Reversalchecks
  - Codierungschecks
  - Plausibilitätschecks
  - Strukturchecks
  - Diagnosechecks
  - Algorithmische Checks
- Kategorien sind nicht orthogonal und einzelne Methoden nicht immer klar zuzuordnen

Replikationschecks

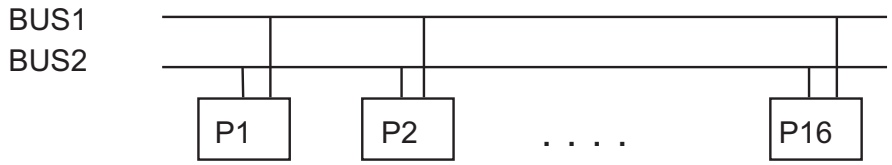
- Alles wird mehrmals gemacht
- Gründlich, aber teuer
- Varianten
  - Identische Replikation
  - Verschiedene Designsifferent designs
  - Wiederholte Ausführung
  - Vergleich mit Standardausführung ➡ **Diagnosechecks**



Timing Checks

- Überprüfung, ob Zeitbedingungen eingehalten werden
- Varianten
  - Extra Zeitüberwachungseinheit (Watchdog)
  - Passive gegenseitige Überwachung
  - Aktive gegenseitige Überwachung

Beispiel Tandem-Computer: „I'am alive" im Sekundentakt, „Are you okay?" zweisekündlich



Reversalchecks

- Ausgaben hängen (in der Regel) deterministisch von Eingaben ab
- Berechnung der Eingaben aus erhaltenen Ausgaben
- Vergleich gegen Eingabe
- Beispiele
  - Wiedereinlesen nach Schreiben
  - Mathematische Funktionen, wie
    - $(\sqrt{x})^2 \stackrel{!}{=} x$
    - $\mathbf{A} \cdot \mathbf{A}^{-1} \stackrel{!}{=} \mathbf{I}$

Coding Checks

- Redundante Datendarstellung
- Beispiele:
  - Paritätsbit ➡ gerade/ungerade
  - Berger-Code ➡ Anzahlen von 0 (oder seltener 1)
  - Checksumme ➡ Addition von Elementen eines Blocks
  - Hamming-Code ➡ Vergrößerung der Hamming-Distanz
  - Cyclic Redundancy Check ➡ Ausnutzung des Lemma von Bézout (Restwerttheorem)
- Mehr im nächsten Abschnitt zu Speicher...

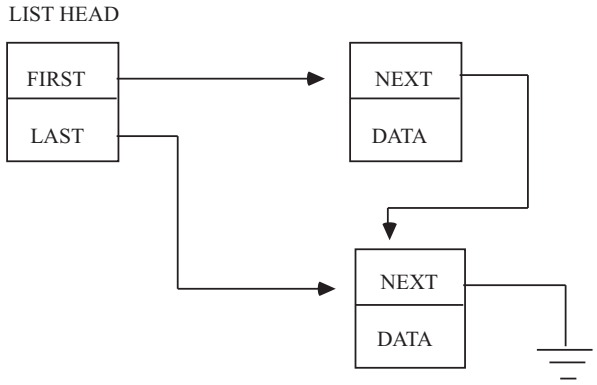
Plausibilitätschecks

- Nutzung des gesunden Menschenverstands
- Ausnutzung von Wissen über das interne Design und Strukturen
- Beispiele:
  - Bereichschecks (z.B. $0° \leq \alpha < 360°$, Arrayindex im vorgegeben Bereich)
  - Konsistenzüberprüfungen (z.B. Wurde jede Eingabe verarbeitet?)
  - Typenchecks (Ist Ergebnis Integer?)

⋮

Strukturchecks

- Überprüfung der Konsistenz von Datenstrukturen oder Systemstrukturen
- Beispiele
  - Anzahl der Elemente
  - Redundante Pointer



  - Liste der PnP-Geräte

Diagnosechecks

- Überprüfung, ob für eine bekannte Menge von Eingaben die korrekten Ausgaben erfolgen
- Typischer Einsatz in Hardwaretestprogrammen (z.B. Power-on self-test, POST)
- Beispiele:
  - Speichertests (später noch diskutiert)
  - Exceptiontests
  - Belastungstests

Algorithmische Checks

- Überprüfen, ob Invarianten eines Algorithmus unberührt bleiben
- Beispiele:
  - Sortierung: Anzahl der Einträge, Checksumme
  - Checksumme bei Matrizenmultiplikation

$$\left[\begin{array}{cc} 1 & 2 \\ 3 & 5 \\ \hline 4 & 7 \end{array}\right] \times \left[\begin{array}{cc:c} 2 & 4 & 6 \\ 1 & 3 & 4 \end{array}\right] = \left[\begin{array}{cc:c} 4 & 10 & 14 \\ 11 & 27 & 38 \\ \hline 15 & 37 & 52 \end{array}\right]$$

7.3 Codingchecks: Hauptspeicher

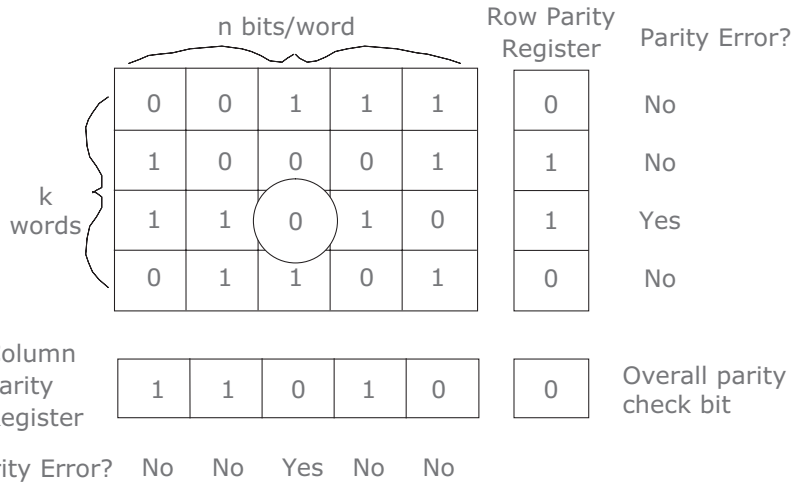
Motivation

- Speicher kommt in Rechner in großen Mengen vor
- Ein einzelner Ein-Bit-Fehler kann zu Systemausfall führen
- Situation verschlimmert sich mit wachsender Kapazität und sinkender Strukturgröße
- ➡ Effiziente Tests notwendig
- **Typische Ansätze:** Codes und Checksummen (die ja auch einen Code darstellen)
- Zwei Anwendungsmethoden von Tests: offline (Produktions- oder Startzeit) oder online

Designkriterien

- **Abdeckung (Coverage)**
  - Gesamtabdeckung
  - Abdeckung bezgl. gegebener Feherarten
- **Overhead**
  - Hardware (zusätzliche Schaltungen, zusätzlicher Speicher)
  - Software
  - Laufzeit (Overhead für Kodierung und Dekodierung)
  - # Checkbits
- **Anwendungsfall**
  - Erkennung
  - Lokalisierung
  - Korrektur

Beispiel: Zweidimensionale Parität



- Erkennt alle 1-Bit-Fehler, alle 2- und 3-Bit-Fehler in k Worten, sowie viele weitere
- Lokalisiert alle 1-Bit-Fehler in k Worten

BERGER-Code

- Anzahl von 1 wird Datenwort hinzugefügt
- k Informationsbits benötigen $\lfloor \log_2 k + 1 \rfloor$ Checkbits
- 100% coverage for single errors
- Coverage calculation tricky for double and other errors
- Low overhead

HAMMING-Codes

- Fehlererkennung und -korrektur
- Beispiel: $(n, k) = (7, 4)$
  7 Bits inklusive 3 Checkbits
- **Hammingdistanz** zwischen zwei Codeworten:
  Anzahl der Unterschiede zwischen korrespondierenden Bitstellen
  $H_d(1001001, 1100101) = 3$
- **Hammingdistanz eines Codes:** minimale Hammingdistanz zweier (unterschiedlicher) Codewörter dieses Codes
- Eine Distanz H_d erlaubt, bis zu $H_d - 1$ Bitfehler zu entdecken
  oder
  bis zu $\lfloor \frac{H_d - 1}{2} \rfloor$ Bitfehler zu korrigieren
- Typisch: $H_d = 3$

Konstruktion von HAMMING-Codes

Konstruktion nach R.W. HAMMING, 1950

- Jede j. Stelle, $j = 2^{i-1}$ mit $i = 1, \ldots, k$ ist ein **Checkbit** (Paritätsbit) c_i.
- Die übrigen Bits sind Datenbits d_l mit $l = 1, \ldots, m$
  - **Beispiel** für $(7, 4)$: $d_4, d_3, d_2, c_3, d_1, c_2, c_1 = h_7, h_6, h_5, h_4, h_3, h_2, h_1$
- Jedes Checkbit bildet eine Parität über eine Anzahl Bits
- **Bildungsregel:** $c_j = h_{2^{j-1}}$ wird für alle Bits mit $(i \mod 2^j) \geq 2^{j-1}$ genutzt (i bezieht sich auf h_i)
  - **Beispiel** für $(7, 4)$:
    - Parität von h_1, h_3, h_5, h_7 ➡ $c_1 = d_1 \oplus d_2 \oplus d_4$
    - Parität von h_2, h_3, h_6, h_7 ➡ $c_2 = d_1 \oplus d_3 \oplus d_4$
    - Parität von h_4, h_5, h_6, h_7 ➡ $c_3 = d_2 \oplus d_3 \oplus d_4$

Beispiel (7,4)-HAMMING-Codes

- Beispiel für einen $(7, 4)$-Hamming-Code
- Keine zwei Codewörter (dritten Spalte) haben eine Hamming-Distanz kleiner 3

Wert	binär	Hamming
0	0000	0000000
1	0001	0000111
2	0010	0011001
3	0011	0011110
4	0100	0101010
5	0101	0101101
6	0110	0110011
7	0111	0110100
8	1000	1001011
9	1001	1001100
10	1010	1010010
11	1011	1010101
12	1100	1100001
13	1101	1100110
14	1110	1111000
15	1111	1111111

Generatormatrix

- HAMMING-Code ist ein **linearer** Code
  ➡ er kann mit einer Generatormatrix erzeugt werden, so dass $h = c \cdot G$
  (Multiplikation ist Modulo 2)
- Beispiel für (7,4)-Code: $G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$
- **Beispiel:** $M = 3$, $c = (0011)$, $h = d \cdot G = (0011110)$

Paritätsmatrix

- Die **Paritätsmatrix** P gibt eine Matrizenform für die Paritätsbildung.
- Beispiel für (7,4)-Code: $P = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- Jede Spalte von P entspricht einer der Paritäten laut der Code-Bildungsregel
- Anmerkung: $G \cdot P = 0$ (wiederum: Multiplikation modulo 2)

Überprüfung von HAMMING-Codes

- Ein empfangenes Codewort h wird überprüft durch Multiplikation modulo 2 mit der Paritäts-Matritze
- Den entstehenden Vektor S nennt man **Syndrom**
- $h \cdot P = S$
- Ist das Syndrom ein Null-Vektor, so ist die Übertragung fehlerlos (wenn die Fehlerannahme zutrifft)
- Bei einem Einzelfehler enthält das Syndrom die fehlerhaften Bitstelle

Fehlerfreier Fall

- Beispiel 1: kein Fehler
  - $d = 1101, h = H(d) = 1100110$
  -

$$S = h \cdot P$$

$$= \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}$$

Fehlerfall

- Beispiel 2: Fehler
  - $d = 1101, h = H(d) = 11\underline{1}0110$
  -

$$S = h \cdot P$$

$$= \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}$$

  - The syndrom equals $(1\,0\,1)$ meaning that the fault is at position 5 of h (counted from right)

Parity and Complement

- Parität kann üblicherweise nur Fehler feststellen, nicht korrigieren
- Mit einem algorithmischen „Trick“ geht es doch
- **Idee**: Schreiben des Komplements auf die gleiche Adresse zum Fehler zu korrigieren
- **Fehlermodell:** max. 1 Bit stuck-at-X
- Beispiel:

1^{st} write	1 1 0 1 0 0 1 1 0	Originaldatum
1^{st} read	1 1 0 1 0 1 1 1 0	Paritätsfehler
$D \to \bar{D}$	0 0 1 0 1 0 0 0 1	Datenkomplement
2^{nd} write	0 0 1 0 1 0 0 0 1	Datenkomplement
2^{nd} read	0 0 1 0 1 1 0 0 1	Paritätsfehler
$D \to \bar{D}$	1 1 0 1 0 0 1 1 0	Komplement (korrigiertes Datum)

7.4 Diagnose auf Systemebene

Systemdiagnose

Bei der **Systemdiagnose** (*system-level diagnosis* oder einfach *system diagnosis*) wird versucht, durch gegenseitiges Überprüfen von Verarbeitungseinheiten festzustellen, wer korrekt und wer fehlerhaft ist.

- Es existiere ein Test (Knoten A testet Knoten B) der eine Aussage „korrekt“ oder „fehlerhaft“ zurückgibt
- **Ziel**: Als defekt erkannt Knoten sollen nicht mehr verwendet werden
- **Problem**: Falschaussagen defekter Knoten über andere Knoten
- **Fragestellungen**
  - (Wann) kann eine Lösung existieren? (Charakterisierung)
  - Wie sieht die Lösung aus?

t-Diagnostizierbarkeit und Syndrom

- Ein System heißt **t-diagnostizierbar**, wenn für jede beliebige Verteilung von bis zu t Fehlern im System jeder dieser Fehler erkannt und lokalisiert werden kann
- **Annahme**: Es gibt einen externen Beobachter, der die Ergebnisse der Tests „einsammeln“ und bewerten kann
- Die Menge der Ergebnisse wird **Syndrom** genannt
- Ein System ist genau dann t-diagnostizierbar, wenn es zu jeder Fehlerverteilung mit bis zu t Fehlern **unterscheidbare** Syndrome gibt
- **Anmerkung**: In Systemen, die mit Zeit arbeiten, werden auch andere Formelzeichen benutzt, z.B. f-Diagnostizierbarkeit

PMC-Modell

- PMC-Modell von PREPARATA, METZE und CHIEN, 1967
- Folgende Annahmen über Testausgänge gelten:

Tester	Testkandidat	Testausgang
korrekt	korrekt	korrekt
korrekt	fehlerhaft	fehlerhaft
fehlerhaft	korrekt	unbestimmt
fehlerhaft	fehlerhaft	unbestimmt

- Zur Vereinfachung sei ein Testausgang „korrekt“ mit 0 bezeichnet und ein Testausgang „fehlerhaft“ mit 1

Diagnostizierbarkeit im PMC-Modell

- Unter der Bedingung, dass sich keine zwei Knoten jeweils gegenseitig testen, gilt

Theorem 7.1

Ein System ist dann t-diagostizierbar, wenn

- $n \geq 2t + 1$
- *Jeder Knoten wird wenigstens von t anderen Knoten getestet*

Diagnostizierbarkeit im PMC-Modell (Forts.)

- Allgemeiner Fall:

Theorem 7.2

Ein System $G(V, E)$ ist genau dann t-diagostizierbar, wenn

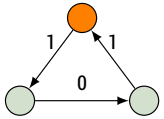
- $n \geq 2t + 1$
- $\forall v \in V : |\Gamma^{-1}(v)| \geq t$
- $\forall p \in \mathbb{N}, 0 \leq p < t, \forall X \subseteq V, |X| = n - 2t + p \rightarrow |\Gamma(X)| > p$

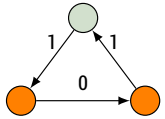
- $\Gamma(Z)$: Menge der von den Knoten aus Z getesteten Knoten,
  $\Gamma^{-1}(Z)$: Menge der Tester von Knoten aus Z

Beweis von Theorem 7.1

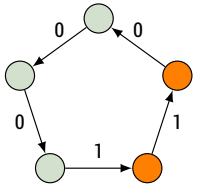
Notwendigkeit. Um die Notwendigkeit zu beweisen, reicht jeweils ein Gegenbeispiel

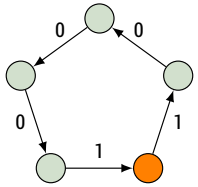
- Notwendigkeit von $n \geq 2t + 1$:





- Notwendigkeit von $|\Gamma^{-1}(v)| \geq t$:





Beweis von Theorem 7.1 (Forts.)

Hinlänglichkeit. Annahme des Gegenteils: $\mathcal{S}$ sei System mit

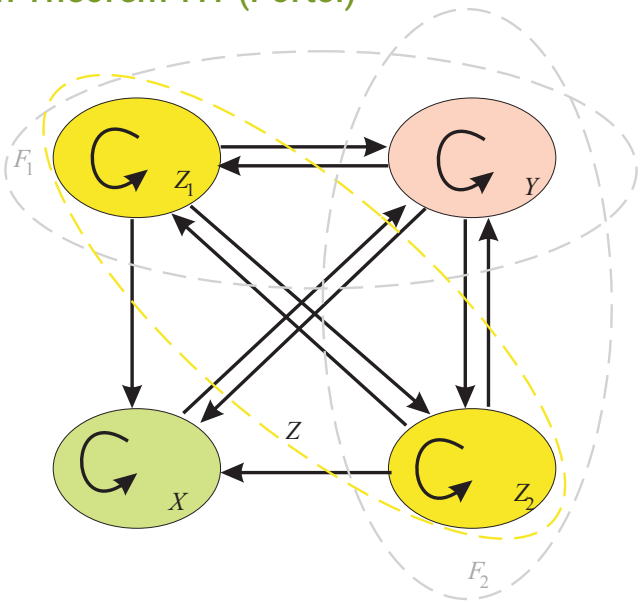
$$n \geq 2t + 1 \quad \text{(A.1)}$$

$$|\Gamma^{-1}(v)| \geq t \quad \text{(A.2)}$$

- $\mathcal{S}$ sei nicht t-diagnostizierbar, ➡ es gibt zwei unterschiedliche Fehlermengen F_1 und F_2, die zum gleichen Syndrom S führen

Wenn F_1 und F_2 ununterscheidbar sein sollen, darf **kein** Element, das **nur** in F_1 **oder** F_2 enthalten ist, von einem Element getestet werden, das in $V \backslash (F_1 \cup F_2)$ (also immer fehlerfrei) ist

Beweis von Theorem 7.1 (Forts.)



Beweis von Theorem 7.1 (Forts.)

Notationen:

- $Y = F_1 \cap F_2$ (Menge der immer fehlerhaften Elemente)
- $Z_1 = F_1 - Y$, (Menge der nur in F_1 fehlerhaften Elemente)
- $Z_2 = F_2 - Y$ (Menge der nur in F_2 fehlerhaften Elemente)
- X sei die Menge der in beiden Fällen korrekten Elemente, $X = V - (F_1 \cup F_2)$

- Sei $E(A, B) = \{(v_1, v_2) \in V | v_1 \in A, v_2 \in B, v_2 \in \Gamma(v_1)\}$, also Tests von Elementen von B aus der Menge A
- $|E(A, B)|$ ist die Anzahl von Test, die Elemente aus A an Elementen aus B durchführen

Beweis von Theorem 7.1 (Forts.)

- Wieviel Tests kann kann es innerhalb einer Menge A von Elementen geben, wenn sich nie zwei Elemente gegenseitig testen ➡ Sektgläserproblem?
- Ein Element kann maximal $|A| - 1$ andere testen, das nächste noch $|A| - 2$, etc.

Lemma 7.1

Wenn sich keine zwei Elemente gegenseitig testen, gilt für jede Menge A:

$$|E(A, A)| \leq \frac{|A| \cdot (|A| - 1)}{2}$$

$$|E(A, B) \cup E(B, A)| \leq |A| \cdot |B| \quad \text{(Lemma 7.1a)}$$
$$\text{wenn } A \cap B = \emptyset$$

Beweis von Theorem 7.1 (Forts.)

Betrachten Tests von Elementen aus Z_1 (laut (A.2)):

$$\begin{aligned} |Z_1| \cdot t &\leq |E(Z_1, Z_1)| + |E(Z_2, Z_1)| + |E(Y, Z_1)| \\ &\leq |E(Z_1, Z_1)| + |E(Z_2, Z_1)| + |Y| \cdot |Z_1| \end{aligned} \quad (1)$$

Analog Tests von Elementen aus Z_2:

$$\begin{aligned} |Z_2| \cdot t &\leq |E(Z_2, Z_2)| + |E(Z_1, Z_2)| + |E(Y, Z_2)| \\ &\leq |E(Z_2, Z_2)| + |E(Z_1, Z_2)| + |Y| \cdot |Z_2| \end{aligned} \quad (2)$$

Beweis von Theorem 7.1 (Forts.)

Addiere (1)+(2):

$$(|Z_1| + |Z_2|) \cdot t \leq |E(Z_1, Z_1)| + |E(Z_2, Z_2)| + \\ + |Y|\,(|Z_1| + |Z_2|) + |E(Z_1, Z_2)| + |E(Z_2, Z_1)|$$

Bei Beachtung von Lemma 7.1 und 7.1a:

$$(|Z_1| + |Z_2|) \cdot t \leq \frac{1}{2}\,(|Z_1|\,(|Z_1| - 1) + |Z_2|\,(|Z_2| - 1)) + \\ + |Y|\,(|Z_1| + |Z_2|) + |Z_1|\,|Z_2|$$

$$2 \cdot t \leq |Z_1| + 2 \cdot |Y| + |Z_2| - 1$$

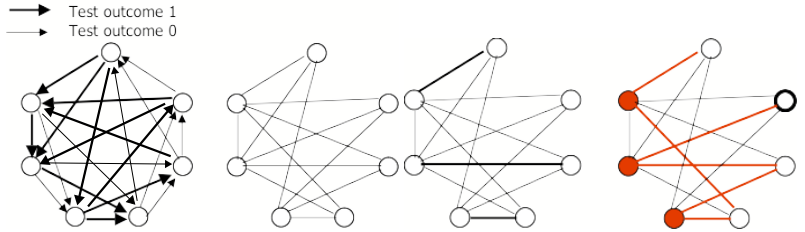
$$2 \cdot t \leq \underbrace{|Z_1| + |Y|}_{=|F_1| \leq t} + \underbrace{|Z_2| + |Y|}_{=|F_2| \leq t} - 1$$

➡ **Widerspruch zu (A.1)** □

Beispiel für Diagnosealgorithmus

Algorithmus von SULLIVAN (modifiziert)

1. Konstruiere den L-Graph (Disagreement-Graph)
   Zwischen v_1 und v_2 existiert eine Kante, wenn aus der Annahme v_1 sei korrekt direkt oder indirekt folgt, dass v_2 fehlerhaft ist
2. Finde ein **maximales Matching** im L-Graph
3. Weise einem Knoten, der **nicht** im Maching ist, den Zustand "korrekt" zu und diagnostiziere von diesen Knoten aus das System



BGM-Modell

- BGM-Modell von BARSI, GRANDONI und MAESTRINI, 1976
- Folgende Annahmen über Testausgänge gelten:

Tester	Testkandidat	Testausgang
korrekt	korrekt	korrekt
korrekt	fehlerhaft	fehlerhaft
fehlerhaft	korrekt	unbestimmt
fehlerhaft	fehlerhaft	fehlerhaft

Diagnostizierbarkeit im BGM-Modell

Notwendige Bedingung für t-Diagnostizierbarkeit im BGM-Modell

Theorem 7.3

Ein System $\mathcal{S}$ sei im BGM-Modell t-diagnostizierbar. Dann gilt:

$$n \geq t + 2$$

Es gibt auch eine allgemeine (hinreichende und notwendige) Bedingung zur t-Diagnostizierbarkeit, aber die ist etwas komplizierter und wird hier ausgelassen

Problemvarianten

- Es gibt eine Reihe weiterer Variation zum Diagnoseproblem, z.B.:
  - **Sequentielle Diagnose:** Einzelne Elemente werden als fehlerhaft erkannt und ersetzt; dann wird die Diagnose fortgeführt
  - **Alternative Diagnosemodelle**
  - **Mengendiagnose:** Es wird eine Menge X, $|X| > t$ ermittelt, die alle fehlerhaften Elemente enthält
  - **Ergebnispropagierung:** Wie müssen die Testergebnisse übertragen werden, wenn sie nicht zentral eingesammelt sondern über die Knoten „geroutet“ werden