IAF0530/IAF9530

Dependability and fault tolerance

Lecture 6
Redundancy (Software and Information)

Gert Jervan
gert.jervan@ati.ttu.ee

Important!

- No lecture on March 28!
- April 4:
  - Draft of the report (by e-mail)
    - Abstract, outline, main references, ca. 1 page
  - Introductory presentation (max. 5 min, max 2-3 slides).

2

Single- and multi-version

- Software fault-tolerance techniques can be divided into two groups:
  - single-version
  - multi-version
- Single version techniques aim to improve fault tolerant capabilities of a single software module
  - fault detection, containment and recovery mechanisms
- Multi-version techniques employ redundant software modules, developed following design diversity rules

3

Single-Version (Dynamic) Techniques

- Dynamic redundancy kicks in only when an error is detected.
- Four phases
  - 1. Error detection:
    fault tolerance techniques effective only when an error is detected
  - 2. Damage assessment and containment:
    to what extent the "damage" has spread because of the delay between a fault and its manifestation/detection?
  - 3. Error recovery:
    techniques to reach from a corrupted to a safe state
  - 4. Fault treatment and continued service:
    error correction.

4

1 - Error Detection

- The goal is to determine that a fault has occurred within a system.
- Various types of acceptance tests are used to detect faults
  - the result of a program is subjected to a test
  - if the result passes the test, the program continues its execution
  - a failed test indicates a fault

5

Acceptance Test

- Acceptance test is most effective if it can be calculated in a simple way and if it is based on criteria that can be derived independently of the program application.
- The existing techniques include
  - timing checks
  - coding checks
  - reversal checks
  - reasonableness checks
  - structural checks
  - replication checks
  - dynamic reasonableness checks

6

2 - Damage Assessment & Containment

- Necessary due to the delay between fault and error
- Goal of containment is to minimize damage caused by a faulty component
  - "firewalling"
- Assessment closely related to containment techniques used
- Techniques for fault containment:
  - modularization
  - partitioning
  - system closure
  - atomic actions

7

3 Fault Recovery

- Once a fault is detected and contained, a system attempts to recover from the faulty state and regain operational status
  - If fault detection and containment mechanisms are implemented properly, the effects of the faults are contained within a particular set of modules at the moment of fault detection.
- The knowledge of fault containment region is essential for the design of effective fault recovery mechanism

8

Recovery

- Forward or Backward
- Forward: continues from an erroneous state by making selective corrections to the system state
  - includes making safe the controlled environment which may be hazardous or damaged because of failure
  - system specific and depends upon accurate predictions
  - e.g., redundant pointers in data structures, self-correcting codes

9

Recovery

- Backward: relies on restoring the system to a previous safe state and executing an alternative section of the program
  - safe functionality but different algorithm
  - the point to which a process is restored is called a recovery point and the act of establishing it is called checkpointing.
  - BER can be used to recover from unanticipated faults including design errors.
  - State restoration is not always possible in (real-time) embedded systems.

10

4 - Fault Treatment and Continued Service

- Even with recovery, the error may recure. Need to eradicate the fault from the system
- Automatic treatment of faults is very application specific
- Make some assumptions. For instance:
  - all faults are transient
- Fault treatment in two stages
  - Fault location
  - System repair
- Fault location
  - use error detection techniques to trace a fault to a component (hardware or software)
  - System repair
    - sometimes it has to be done while the system is in operation.

11

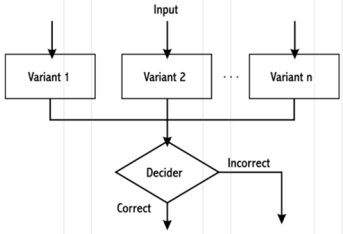
Multi-Version Techniques

- Multi-version techniques use two or more versions the same software module, which satisfy design diversity requirements.
  - different teams, different coding languages or different algorithms can be used to maximize the probability that all the versions do not have common faults

12

Design Diversity

- Higher cost



13

SFT Techniques Using Design Diversity

Techniques	Abbr.	Error Processing
Recovery Blocks	RcB	Error detection by AT and backward recovery
N-Version Programming	NVP	Vote
N Self-Checking Programming	NSCP	Error detection by AT and forward recovery

AT – Acceptance Test

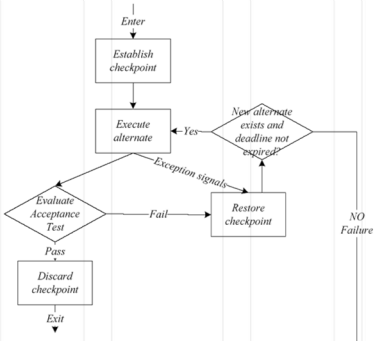
14

Recovery Blocks

- Combines checkpoint and restart approach with standby sparing redundancy scheme
- n different implementations of the same program
  - Only one of the versions is active
  - If an error if detected by the acceptance test, a retry signal is sent to the switch
  - The system in rolled back to the state stored in the checkpoint memory and the execution is switched to another module

15

Recovery Blocks



16

Recovery Blocks

Method	Recovery block
Error Processing Technique	Error detection by AT and backward recovery
Criteria of Accepting Result	Absolute, with respect to specification
Execution Scheme	Sequential
Consistency of Input Data	Implicit, from backward recovery principle

17

Recovery Blocks

- A language level support for backward error recovery
  - blocks in the normal programming language sense, but
  - at the entrance to the block is an automatic recovery point and
  - at the exit an acceptance test to test that the system is an an acceptable state
  - if the acceptance test fails, the program is restored to the recovery point at the beginning of the block and an alternative module is executed
  - repeat this process with alternative modules
  - if all fail, recovery must take place at a higher level
- In terms of four phases of software fault tolerance
  - Error detection <-> acceptance test
  - Damage assessment <-> not needed due to BER
  - Fault treatment <-> stand-by spare code

18

Recovery Blocks

- Similarly to cold and hot standby sparing, different version can be executed either serially, or concurrently
  - Serial execution may require the use of checkpoints to reload the state before the next version is executed
  - The cost in time of trying multiple versions serially may be too expensive, especially for a real-time system.
  - A concurrent system requires n redundant hardware modules, a communications network to connect them and the use of input and state consistency algorithms.

19

Syntax of Recovery Blocks

```
ensure <acceptance test>
by
   <primary module>
else by
   <alternative module>
else by
   <alternative module>
   ...
else by
   <alternative module>
else error
```

- Recovery blocks can be nested
- If all alternatives in a nested recovery block fail the acceptance test, the outer level recovery point will be restored
  - (and an alternative module to that block will be executed).

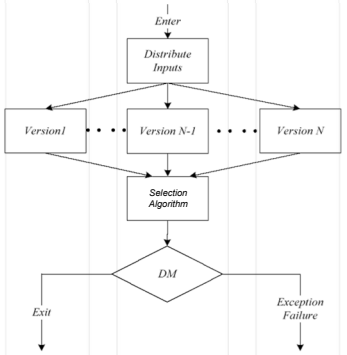
20

N-Version Programming

- Resembles N-modular hardware redundancy
- N different software implementations of a module are executed concurrently.
- The selection algorithm (voter) decides which of the answers is correct
  - a voter is application independent
  - this is an advantage over recovery block fault detection mechanism, requiring application dependent acceptance tests

21

NVP



22

N-version Programming

Method	N-version programming
Error Processing Technique	Vote
Criteria of Accepting Result	Relative, on variant results
Execution Scheme	Parallel
Consistency of Input Data	Explicit by dedicated mechanisms

23

N-Version Programming

- Consists of independent generation of N (>2) functionally equivalent programs from same initial specifications
  - Design Diversity, Different Programming Language, Methods..
- Programs execute concurrently, results are arrived at by consensus (majority voting).
- Questions
  - How are results compared? How is voting conducted?
- NVP depends upon
  - good initial specification, independence of effort, abundance of effort.
- NVP can be taken further
  - compiling, processing, ...

24

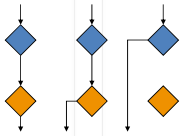
NVP

- Controlled by a driver process
  - invokes each of the versions
  - waiting for the versions to complete
  - comparing and acting on the results
- Problem: assumes programs run to completion!
  - So the versions must actually interact (with the driver program)
    - Comparison Points: points in the versions when programs must communicate their votes to the driver process
    - Defines granularity of the fault tolerance
  - How the versions communicate and synchronize depend upon the programming language used, its model of concurrency

25

Vote Comparison in NVP

- Efficiency of vote comparison is critical
- Complicated by comparison procedure
  - Not all results are single numeric values
  - The "consistent comparison problem"
    - When using "thresholds" for comparison the errors can stack up, resulting different execution paths in *all* versions.



Two sequential thresholding lead to different execution paths in all three versions.

The problem will reappear even when using inexact comparison (just have to be near a threshold value).

And what happens when there are multiple solutions?

26

NVP versus RB

- NVP is static where as RB is dynamic redundancy
- Both have design overheads
  - alternative algorithms
  - NVP requires a driver
  - RB requires an acceptance test
- Runtime overheads
  - NV requires more resources
  - RB requires establishing recovery points
- Both susceptible to errors in requirements
- Error detection
  - vote comparison (NVP) versus acceptance test (RB)
- Atomicity requirement
  - NV vote before it outputs to the environment, RB must output only following the passing of the acceptance test.

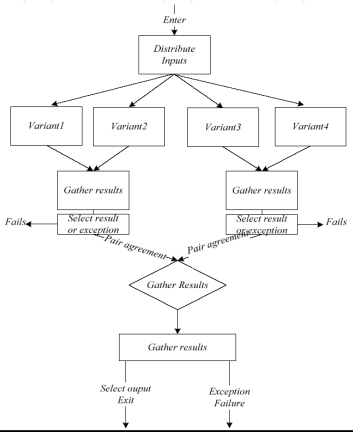
27

N Self-Checking Programming

- N self-checking programming combines recovery block concept with N version programming
- The checking is performed either by using acceptance tests, or by using comparison.
- Examples of applications of N self-checking programming:
  - Lucent ESS-5 phone switch
  - Airbus A-340 airplane

28

NSCP



29

NSCP

Method	N self-checking programming
Error Processing Technique	Error detection and result switching Then, Detection by comparison or by AT(s)
Criteria of Accepting Result	Relative, on variant results or Absolute with respect to specification
Execution Scheme	Parallel
Consistency of Input Data	Explicit, by dedicated mechanisms

30

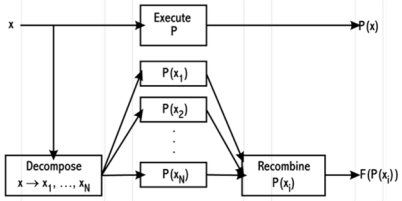
Comparison

- N self-checking programming using acceptance tests
  - The use of separate acceptance test for each version is the main difference of this technique from recovery blocks
- N self-checking programming using comparison
  - resembles triplex-duplex hardware redundancy
  - An advantage over N self-checking programming using acceptance tests is that the application independent decision algorithm is used for fault detection

31

Data Diversity

- To complement design diversity
- Using data re-expression algorithms (DRA) to obtain logically equivalent variants of the input data



Data re-expression via decomposition and recombination

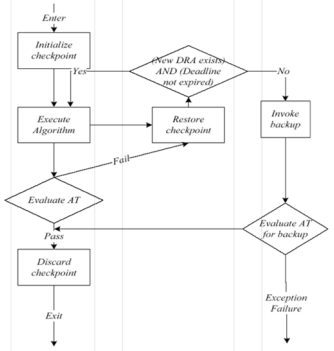
32

SFT Techniques Using Data Diversity

SFT Techniques	Abbr.	Error Processing
Retry Blocks	RtB	Acceptance test and Backward recovery
N-Copy Programming	NCP	Run the same process concurrently or sequentially

33

Retry Blocks



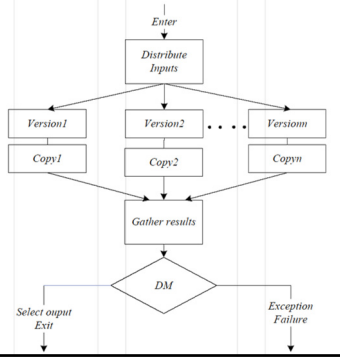
34

Retry Blocks

Method	Retry blocks
Error Processing Technique	Error detection by AT and backward recovery by DRA
Criteria of Accepting Result	Absolute, with respect to specification
Execution Scheme	Sequential
Consistency of Input Data	Implicit, from backward retry principle

35

NCP



36

N-copy Programming

Method	N-copy programming
Error Processing Technique	Decision mechanism (DM) and forward recovery
Criteria of Accepting Result	Relative, on variant results
Execution Scheme	Parallel
Consistency of Input Data	Explicit by dedicated mechanisms

37

Design Diversity

- The most critical issue in multi-version software fault tolerance techniques is assuring independence between the different versions of software through design diversity
- Software systems are vulnerable to common design faults if they are developed by the same design team, by applying the same design rules and using the same software tools

38

Design Diversity

- Decision to be made when developing a multiversion software system include
  - which modules are to be made redundant
    - usually less reliable modules are chosen
  - the level of redundancy
    - procedure, process, whole system
  - the required number of redundant versions
  - the required diversity
    - diverse specification, algorithm, code, programming language, testing technique
  - rules of isolation between the development teams

39

Environment Diversity

- To diversify the software operating circumstance temporarily.
- The typical examples of environment diversity technique are progressive retry, rollback rollforward recovery with checkpointing, restart, hardware reboot, etc.

40

An Adaptive Approach for n-Version Systems

- Model and manage different quality levels of the versions by introducing an individual weight factor to each version of the n-version system.
- This weight factor is then included in the voting procedure, i.e. the voting is based on a weighted counting.

41

Why Fuzzy Voting

- In traditional voting, equality relation regards two real numbers as equal if their difference is smaller than fixed tolerance ε. For different version outputs that are "closer" to each other than the fixed threshold there is no gradual comparison. As a result, certain interconnection of faults could incur incorrect selection.
- Fuzzy equivalence relation results in more reliable systems

42

Fuzzy Equality Equation

- Traditional Equality Equation

$$r_{i,j} = \begin{cases} 1, & if \mid x_i - x_j \mid \leq \varepsilon \\ 0, & otherwise \end{cases}$$

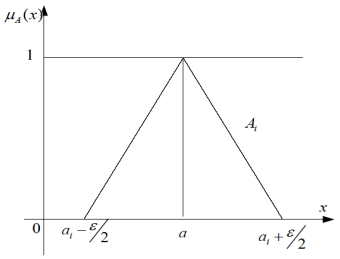
- Fuzzy Equality Equation

$$\mu_{A_i}(x_i) = \begin{cases} 1 - \dfrac{\mid a_i - x_i \mid}{\varepsilon / 2}, & if \mid x_i - a_i \mid \leq \varepsilon / 2 \\ 0, & otherwise \end{cases}$$

43

Output of Fuzzy Sets (Triangular Shape)

- The fuzzy logic maps the input vector into an output nonlinearly



44

Software Aging

- When software application executes continuously for long periods of time, some of the faults cause software appear to age due to the error conditions that accrue with time and/or load. This phenomenon is called software aging which is reported in
  - Telecommunication billing application over time experiences a crash or a hang failure.
  - A telecommunication switching software
  - Netscape and xrn
  - Safety critical systems Patriot missile's software, where the accumulated errors led to a failure that resulted in loss of human lives.

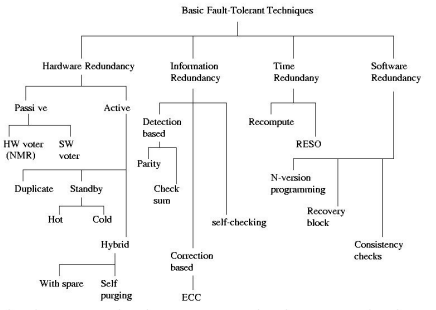
45

Discussion

- Each software fault tolerance technique need to be tailored to particular applications.
- This should also be based on the cost of the fault tolerance effort required by the customer. The differences between each technique provide some flexibility of application.

46

A summary chart of all techniques



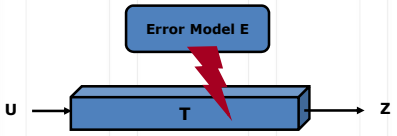
47

Information redundancy

- Definition
  - Information redundancy is the addition of redundant information to data to allow fault detection, fault masking or possibly fault tolerance.
- Error detecting and correcting codes (EDC codes)
  - Encoding of information for transmission in noisy environments
  - Later for dependability: communications, memory, storage, etc.

48

Error Model



- Functional faults
- Technological faults
- Disruptions due to the environment

49

Error Classes

- An error is **single** when it only affects a single bit of the output Z
- An error is **multiple of order p** when it affects at most p bits of Z
- Burst error – the errorneous bits of Z are within an l-distance neighbourhood

50

Code

- **Code of length n** is a set of n-tuples satisfying some well-defined set of rules
- **Binary code** uses only 0 and 1 symbols
  - binary coded decimal (BCD) code
    - uses 4 bits for each decimal digit

```
0000    0
0001    1
0010    2
...
1001    9
```


51

Code Word

- A **code word** is a collection of symbols used to represent a particular piece of data based on specified code
- A **word** is an n-tuple not satisfying the rules of the code
- Codewords should be a subset of all possible 2n binary tuples to make error detection/correction possible
  - BCD: 0110 valid; 1110 invalid
  - any binary code: 2013 invalid
- The number of codewords in a code C is called the **size** of C

52

Encoding vs. decoding

- The **encoding process** is the process of determining the corresponding code word for a particular data item.
  - Example: given the decimal 9, encoding determines the BCD representation of 1001.

data → encoding → code word

- The **decoding process** is the process of recovering the original data from the code word.
  - Example: decoding transforms the BCD code 0011 into the decimal 3

code word → decoding → data

53

Encoding/decoding

- 2 scenario if errors affect codeword:
  - correct codeword → another codeword
  - correct codeword → word

54

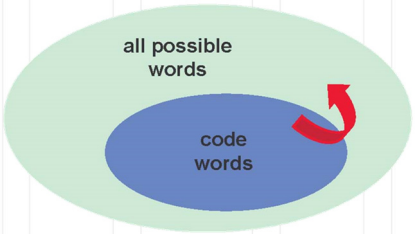
Error Detection

- We can define a code so that errors introduced in a codeword force it to lie outside the range of codewords
  - Basic principle of **error detection**

55

Error Detection

- Error detection: code word is invalid



56

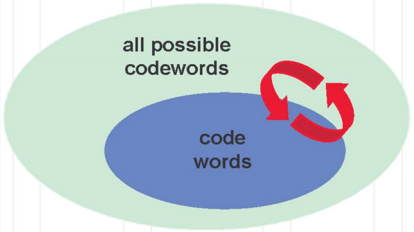
Error Correction

- We can define a code so that it is possible to determine the correct code word from the erroneous codeword
  - Basic principle of **error correction**

57

Error Correction

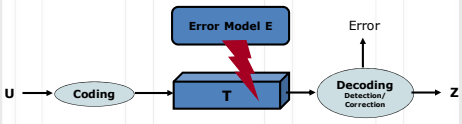
- Error correction: correct word can be identified from the corrupted word



58

EDC/ECC

- Error Detecting and Correcting Codes



- Separable/non-separable codes
  - Separable: original information is appended with new information

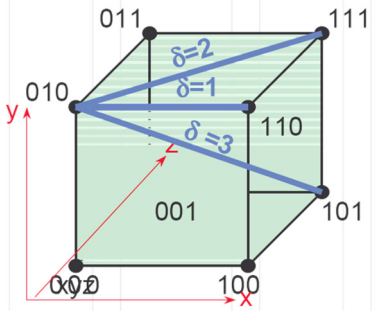
59

EDC/ECC

- Characterized by the number of bits that can be corrected
  - double-bit detecting code can detect two single-bit errors
  - single-bit correcting code can correct one single-bit error
- Hamming distance gives a measure of error detecting/correcting capabilities of a code
  - Number of bit positions in which the two words differ
    - Hamming distance of 1: 0000 to 0001; 2: 0000 to 0101
  - Code distance:
    - Minimum Hamming distance between any two valid code words

60

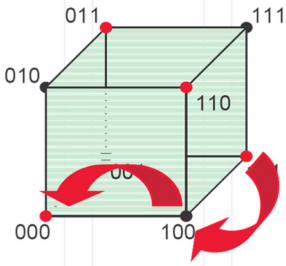
3-dimensional space (3-bit words)



61

Error Detection

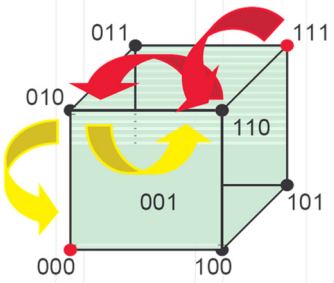
- If codewords are on distance ≥ 2, we can detect single-bit errors



62

Error Correction

- If codewords are on distance ≥ 3, we can correct single-bit errors



63

Code Distance

- Code distance is the minimum Hamming distance between any two distinct codewords

$C_d = 2$ code detects all single-bit errors
code: 00, 11
invalid code words: 01 or 10

$C_d = 3$ code corrects all single-bit errors
code: 000, 111
invalid code words: 001, 010, 100, 101, 011, 110

64

Code Capabilities

- To correct ε-bit errors a code should have the code distance Cd ≥ 2ε +1
- To be able to detect ε-bit errors a code should have the code distance Cd ≥ ε +1
- A code can correct up to c bit errors and detect up to d additional bit errors if and only if:

$$2c + d + 1 \leq Cd$$

65

Separable/non-separable code

- Separable code
  - codeword = data + check bits
  - e.g. parity: 11011 = 1101 + 1
- Non-separable code
  - codeword = data mixed with check bits
  - e.g. cyclic: 1010001 -> 1101
- Decoding process is much easier for separable codes (remove check bits)

66

Information Rate

- The ratio k/n, where
  - k is the number of data bits
  - n is the number of data + check bits

  is called the information rate of the code
- Example: a code obtained by repeating data three times has the information rate 1/3

67

Code Characterization

- Cost: number of bits n that it needs
- Power of expression (cardinality): number of codewords N that it is able to represent
- Error model: defining the errors detected and/or corrected
  - Redundancy rate: rr=r/k (r: added bits)
  - Density of a code: d=N/2n
  - Coverage rate

68

Information redundancy

- Use of parity
  - very effective single error detection
  - encoding and decoding cost is low
  - commonly used in memories, transmission over short reliable channels
  - limitations
    - unable to detect common multiple errors
    - can not be used in data transformation - for example addition does not preserve parity

69

Information redundancy

- Error correcting codes
  - triplication
  - Hamming code
  - byte error detection/correction
  - cyclic code
- m-out-of-n codes
  - encode each word (data/control) such that the coded word is of length n and each coded word has exactly m 1's in it
    - can detect all single errors
    - can detect all unidirectional multiple errors

70

Information redundancy

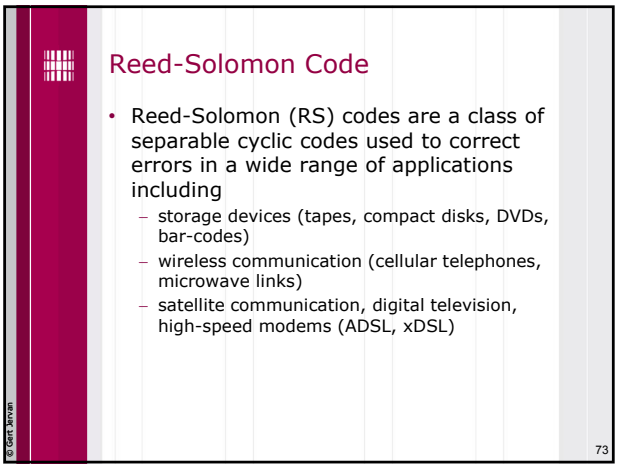
- Berger codes
  - n information bits are encoded into an n+k bit code word. The k check bits are binary encoding of the number of 1's (or 0's) in the n information bits
    - can detect all single errors
    - can detect all unidirectional multiple errors if carefully designed
- Arithmetic codes
  - AN code
    - used for arithmetic function unit designs
    - each data word is multiplied by a constant A
    - makes use of the identity A(N+M) = AN + AM
    - choice of A is important

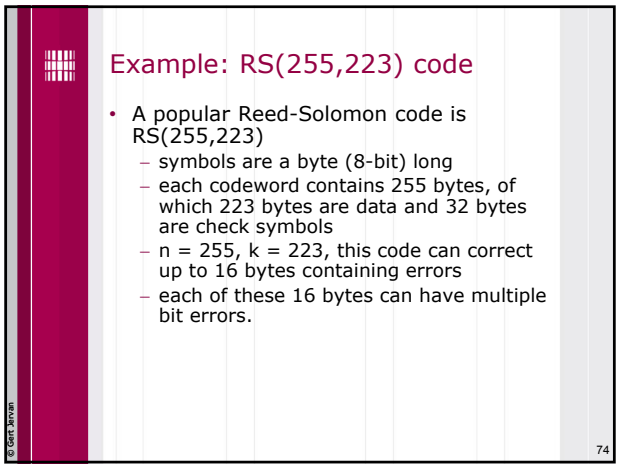
71

Information redundancy

- Arithmetic codes (Contd.)
  - Residue code
    - makes use of the fact (M+N) mod k = (M mod k + N mod k) mod k
  - Checksums
    - data is sent/stored with a checksum and when used the checksum is regenerated and compared to the a priory known checksum
    - functions used for checksum
      - add, exclusive-OR (bit wise), end with end around carry, LFSR, …
    - limitation
      - can only perform (normally) error detection

72


Reed-Solomon Code
• Reed-Solomon (RS) codes are a class of separable cyclic codes used to correct errors in a wide range of applications including
– storage devices (tapes, compact disks, DVDs, bar-codes)
– wireless communication (cellular telephones, microwave links)
– satellite communication, digital television, high-speed modems (ADSL, xDSL)
© Gert Jervan
73


Example: RS(255,223) code
• A popular Reed-Solomon code is RS(255,223)
– symbols are a byte (8-bit) long
– each codeword contains 255 bytes, of which 223 bytes are data and 32 bytes are check symbols
– n = 255, k = 223, this code can correct up to 16 bytes containing errors
– each of these 16 bytes can have multiple bit errors.
© Gert Jervan
74