Software Fault Tolerance
Independent Concurrent Systems

Jörg Kienzle
Software Engineering Laboratory
School of Computer Science
McGill University

Overview

- Design Diversity (Pullum 2.2)
- N-Version Programming (Pullum 4.2)
- Voting (Pullum 7.1)
  - Similarity
  - Consistent Comparison Problem
  - Exact Majority Voter, Mean Voter, Median Voter, Consensus Voter, Formal Majority Voter
- N-Copy Programming (Pullum 5.2)

Design Diversity Idea

- Identical copies (replicates) of software can not increase reliability in the presence of software design faults

  ⇒ Design diversity:

  Provision of identical services through separate design and implementations
- Components providing identical functionality are called versions, variants, alternatives, modules

Design Diversity Process

- Establish initial specification
  - Functional requirements
  - Decision (adjudication) points
    - Data per se, and data format to be compared
- Possible to provide diverse specifications
  - + different inputs ⇒ functional diversity
- Each developer / development organization implements a variant that provides the required output

Design Diversity Goals & Issues

- Make versions as diverse and independent as possible
  - Low probability of common-mode failures: Variants should fail on disjoint subsets of the input space
  - High reliability: At least one variant should be operational all times
- Lack of diversity in variants might lead to similar errors occurring at the same decision point

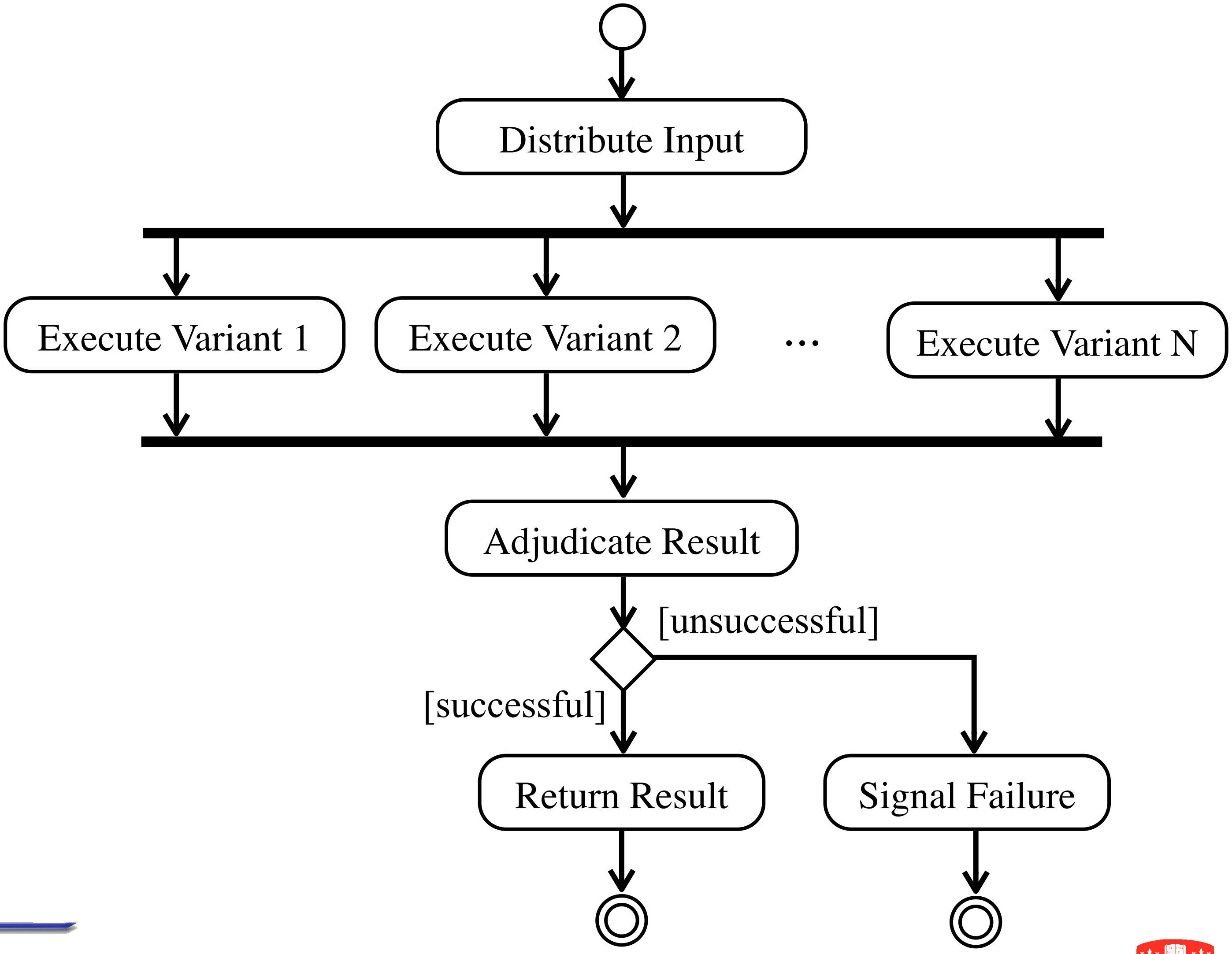
N-Version Programming (1)

- Suggested in 1972 [Elm72], developed by Avizienis and Chen [CA78]
- N (at least 2) versions run in parallel
- A decision mechanism selects the "best" result
- Design diverse, static technique
  (versions are executed regardless of which result will be finally used)
- N-version programming can be seen as the concurrent version of recovery blocks

N-Version Programming (2)

```
run Version 1 .. Version n in parallel
if Decision Mechanism
  (Result 1, .. Result n) return Result
else signal failure exception
```

Parallel Design Diversity Concept



N-Version Programming Discussion

- Runs in a multiprocessor environment
- Small run-time overhead
  - Time of the slowest version
  - Running the decision algorithm
  - Synchronization
- Continuity of service
- Possible to use results of the versions to perform back-to-back testing

Voting on an Outcome

- Voters or decision makers compare the results of two or more versions and decide on the correct result, if one exists
  - Two version voters are also called comparators
- Voters tend to be single points of failure
  - Highly reliable
  - Effective
  - Efficient
- Voters face several fundamental problems

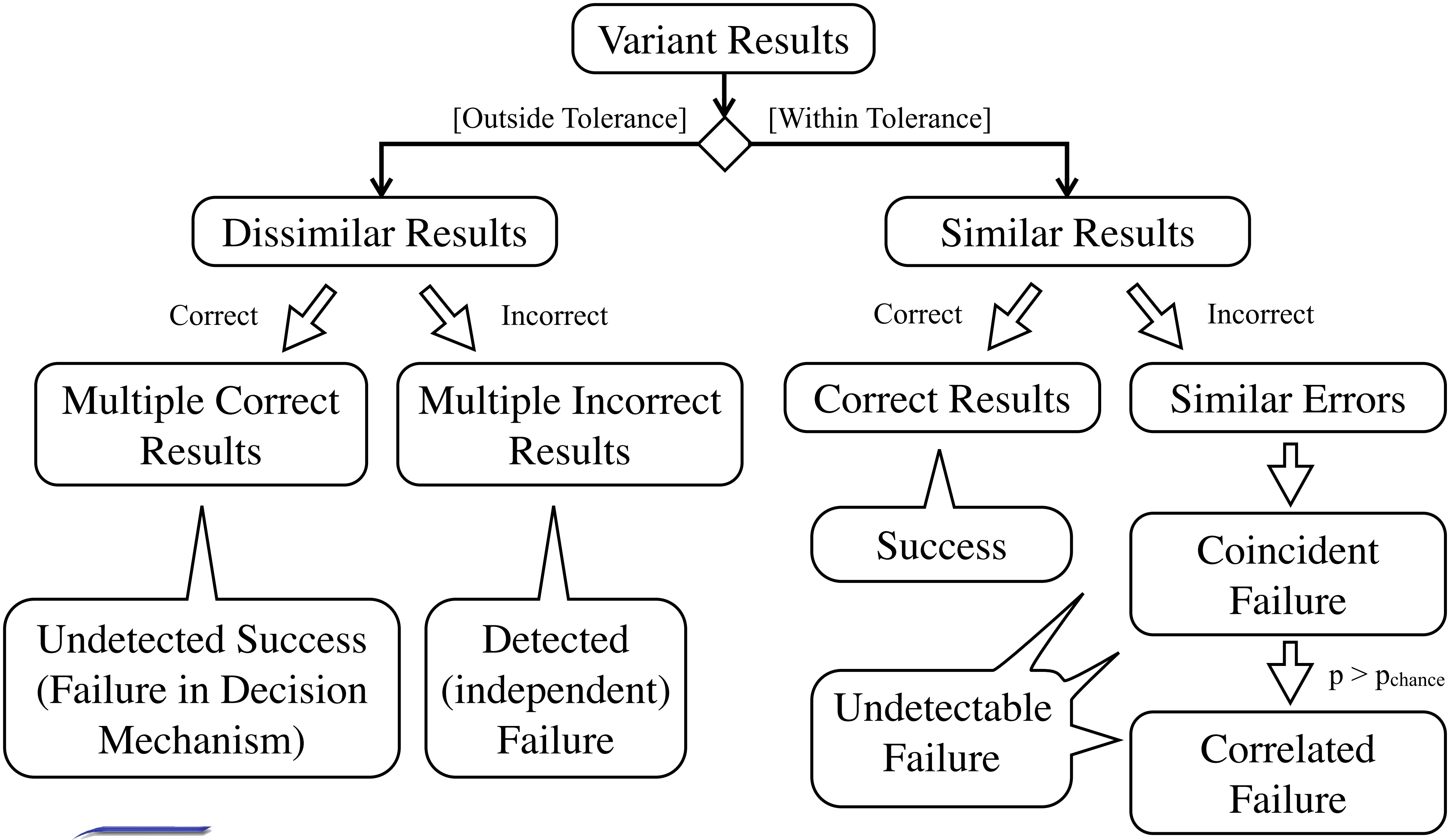
Similarity

- Similar results (approximately equal, within a specified tolerance)
  - Use of floating-point arithmetic
  - Diverse algorithms
- Problem for adjudication
  - Decision mechanism must be tolerant
- Similar incorrect results that are considered correct are called similar errors (or identical wrong answers)

Similarity Definitions (2)

- Coincident failure: Multiple variants fail on the same input case [EL85]
- Correlated failures (or dependent failures): The actual, measured probability of coincident variant failures is different from what would be expected by chance occurrence of these failures [LM89]
- Multiple correct results: Two or more correct answers exist for an algorithm for the same input
  - Example: finding roots of an n-th order equation

Taxonomy of Variant Results



Variant Results

[Outside Tolerance]

[Within Tolerance]

Dissimilar Results

Similar Results

Correct

Incorrect

Correct

Incorrect

Multiple Correct Results

Multiple Incorrect Results

Correct Results

Similar Errors

Success

Coincident Failure

Undetected Success (Failure in Decision Mechanism)

Detected (independent) Failure

$p > p_{chance}$

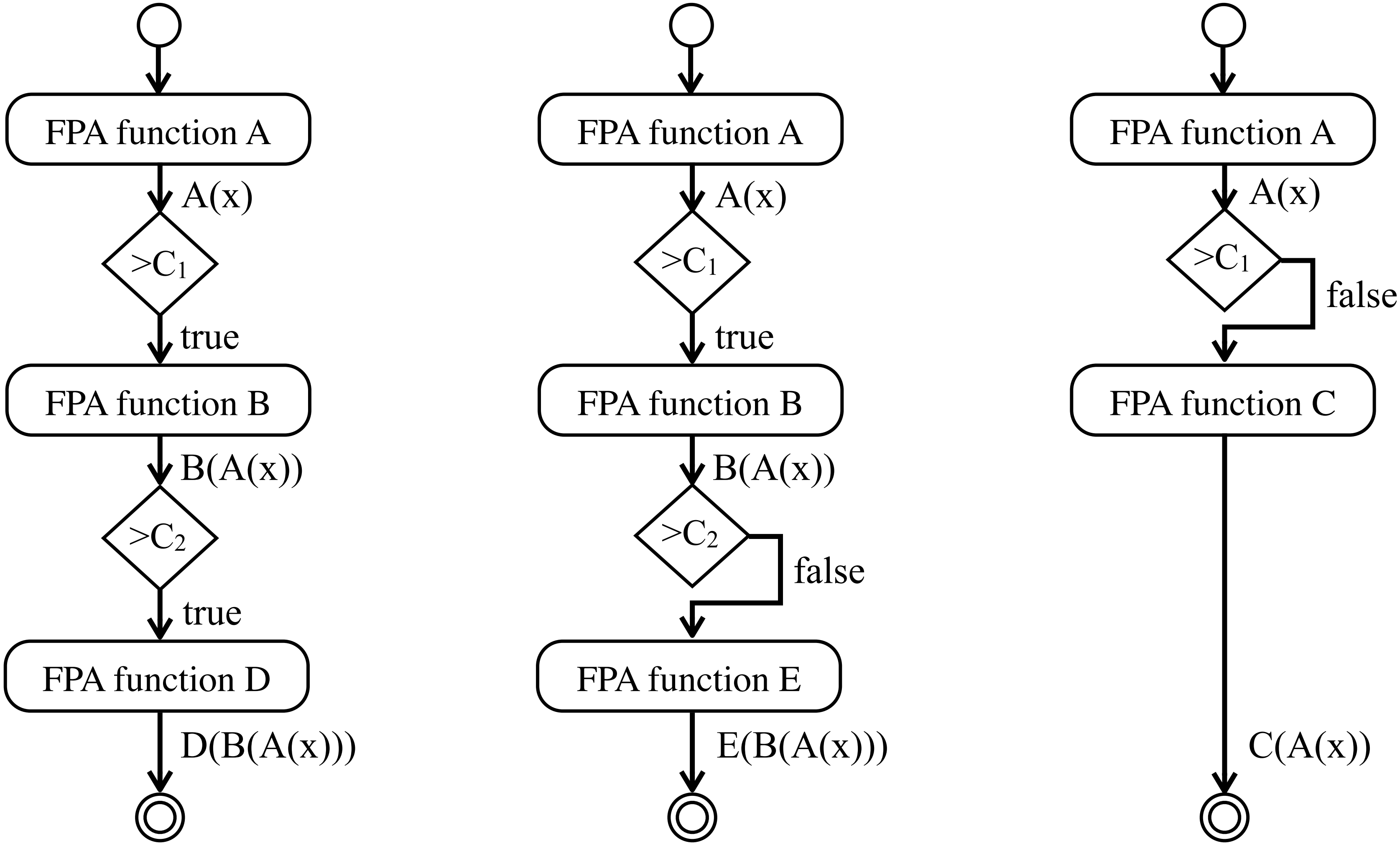
Undetectable Failure

Correlated Failure

Consistent Comparison Problem (1)

- Whenever the specification of a problem requires to make comparisons, it is not possible to guarantee that variants will make the same decision [BKL87]
  - Use of floating-point arithmetic
  - Diverse algorithms (different execution paths)
- May lead to output values that are completely different!

Consistent Comparison Problem (2)



Consistent Comparison Problem (3)

- Specifications do not (and probably cannot) describe required results down to the bit level for every computation and every input
- Without communication between the variants, there is no solution to the consistent comparison problem [BKL87]
  - Approximate comparison / rounding does not help
  - Exact arithmetic impractical

Consistent Comparison Problem (4)

- N-version systems have a non-zero probability of being unable to reach consensus
  ⇒ introduce additional faults!
- Not always a problem, e.g. in systems with no history (e.g. simple control systems)
  - Transient phenomenon (single-cycle failure)
  - Avoidance using confident signals (send an additional confidence value to the adjudicator)

Consistent Comparison Problem (5)

- Systems with state
  - Failure to reach consensus may depend on differences in internal state
- Systems with convergent states
  - State information revised over time
    - State will eventually become consistent again
  - Example:
    Avionics, height above ground determines flight mode
  - Again, confident signals may help
- Systems with non-convergent states
  - Inconsistency may persist forever
  - Only solution: revert to a backup system

Developing a Voter

- Make it as simple as possible (but not simpler :)
  - Complex voters are error-prone
- Write reusable (technique independent) decision makers
- Write fault-tolerant decision makers
  - Distributed voting (requires consensus algorithms)
- When testing your system, test the voter as well!

When is it a Good Time to Vote?

- Coarse Granularity
  - Comparisons are performed infrequently or at the level of complex data types
  - Reduces overhead
  - Increases the amount of possible diversity among variants, which might make decision more difficult
- Fine Granularity
  - Comparisons are performed frequently or at the basic data level
  - High overhead
  - Decreases the possibility for diversity

Exact Majority Voter [Avi85]

- Select the value of the majority of variants
- M-out-of-N voter
  - N often = 3
  - M = $\lceil (n+1)/2 \rceil$

Results of variants	(A, A, A)	(A, A, B) (A, B, A) (B, A, A)	(A, A, ∅) (A, ∅, A) (∅, A, A)	(A, B, C)	(Other)
Voter Result	A	A	Exception	Exception	Exception

Mean Voter

- Select the mean or weighted average of the results provided by the variants
- Can only be used on numeric output values
- Can use weights based on the trustworthiness of variants (obtained from confidence signals, or updated based on previous results, etc.)

Results of variants ($A<B<C$)	(A, A, A)	(A, A, B) (A, B, A) (B, A, A)	$(A, A, \varnothing)$ $(A, \varnothing, A)$ $(\varnothing, A, A)$	(A, B, C) (C, B, A) (A, C, B) …	(Other)
Voter Result	A	Mean(A,A,B) Mean(w_1A,w_2B, w_3C)	Exception	Mean(A,B,C) Mean(w_1A,w_2B, w_3C)	Exception

Voter Discussion (1)

- Exact majority voter
  - Works well for discrete (integer or binary) results
  - Assumes one correct output for each function
  - Is defeated by MCR
  - Is defeated by FPA variations
  - Can’t handle approximate DRAs
  - Does not have to wait for all versions, only until a majority can be established
- Mean voter
  - Good when the probability of correctness decreases with increasing distance from the ideal result [GS90]
  - Is vulnerable to MCR
  - Handles FPA variations well
  - Works well with approximate DRAs

Consensus Voter [V93]

- Generalization of the majority voter
- Find the biggest set (#elements ≥ 2) of matching results
  - If N = 3, then the consensus voter is equivalent to the exact majority voter

Results of variants	(A, B, B, B, C)	(A, B, B,C, D)	(A, A, B, C, C)	(A, B, C, D, E)	(with ∅)
Voter Result	B (maj.)	B (unique agreement)	A or C (tie agreement)	Exception	Exception

Median Voter

- Select the median of the results provided by the variants
- Can only be used on “ordered” values
- Assumption: no incorrect result lies between two correct results

Results of variants (A<B<C)	(A, A, A)	(A, A, B) (A, B, A) (B, A, A)	(A, A, ∅) (A, ∅, A) (∅, A, A)	(A, B, C) (C, B, A) (A, C, B) …	(Other)
Voter Result	A	A	Exception	B	Exception

Voter Discussion (2)

- Median voter
  - Not defeated by MCR
  - Outperforms exact majority and mean voters [BS90]
  - Handles FPA variations well
  - Works well with approximate DRAs
- All previous schemes have problems when a version produces no results
  - Idea: use dynamic voters, e.g. only take into account the results of versions that are available after a given time
    - The reason why no result might be available include crash failures, or ommision, or timing failures of one or multiple variants

Dynamic Majority Voter

- Select the value of the majority of variants that have produced a result

Results of variants	(A, A, A)	(A, A, B) (A, B, A) (B, A, A)	(A, A, ∅) (A, ∅, A) (∅, A, ∅) ?	(A, B, C)	(A, B, ∅)
Voter Result	A	A	A	Exception	Exception

Comparison Tolerances

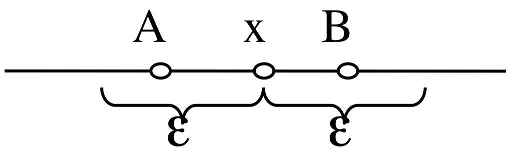
- To handle FPA variations, comparison tolerances can be added
  - Works well with the exact majority or consensus voter ⇒ formal majority or formal consensus voter (sometimes also called tolerance voter or inexact voter)
  - Define ε, i.e. the maximum distance allowed between two correct output values for the same input value
  - Calculate all “distances”
    - $|A - B| = \delta 1$
    - $|A - C| = \delta 2$
    - $|B - C| = \delta 3$

Tuning ε

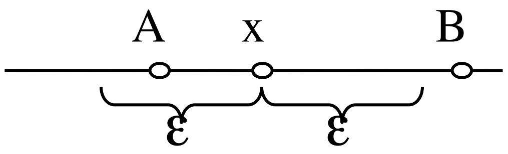
- If ∀i: δi ≤ ε, then there exists an agreement event, otherwise there exists a conflict event
- When a majority of variants produce an acceptable result, then there is a no failure event, otherwise, there is a failure event
- Good situations
  - No failure occurs with agreement
  - Failure occurs with conflict
- Bad situations
  - No failure occurs with conflict: false alarm ⇒ the tolerance ε is probably too small
  - Failure occurs with agreement: undetected failure ⇒ the tolerance ε is probably too big

Formal Majority Voter

- Select the value of the majority of variants using a tolerance of ε
- Select one output x, then construct the feasibility set FS including all results that are within the tolerance ε
  - If FS contains at least a majority of results, then randomly select one of them



Result: A or x or B



Result: A or x

Which Voters are Best?

- If safety is the primary concern
  - Exact majority voter, formal majority voter, dynamic majority voter
  - Rather raise an exception and present no output instead of trying to guess the correct one
- If an answer is better than no answer, i.e. reliability is the primary concern
  - Median voter, mean voter, weighted average voters
  - Always reach a decision (unless they fail themselves)
- There are many more voters tailored to specific application areas, sometimes also combining ideas taken from acceptance tests

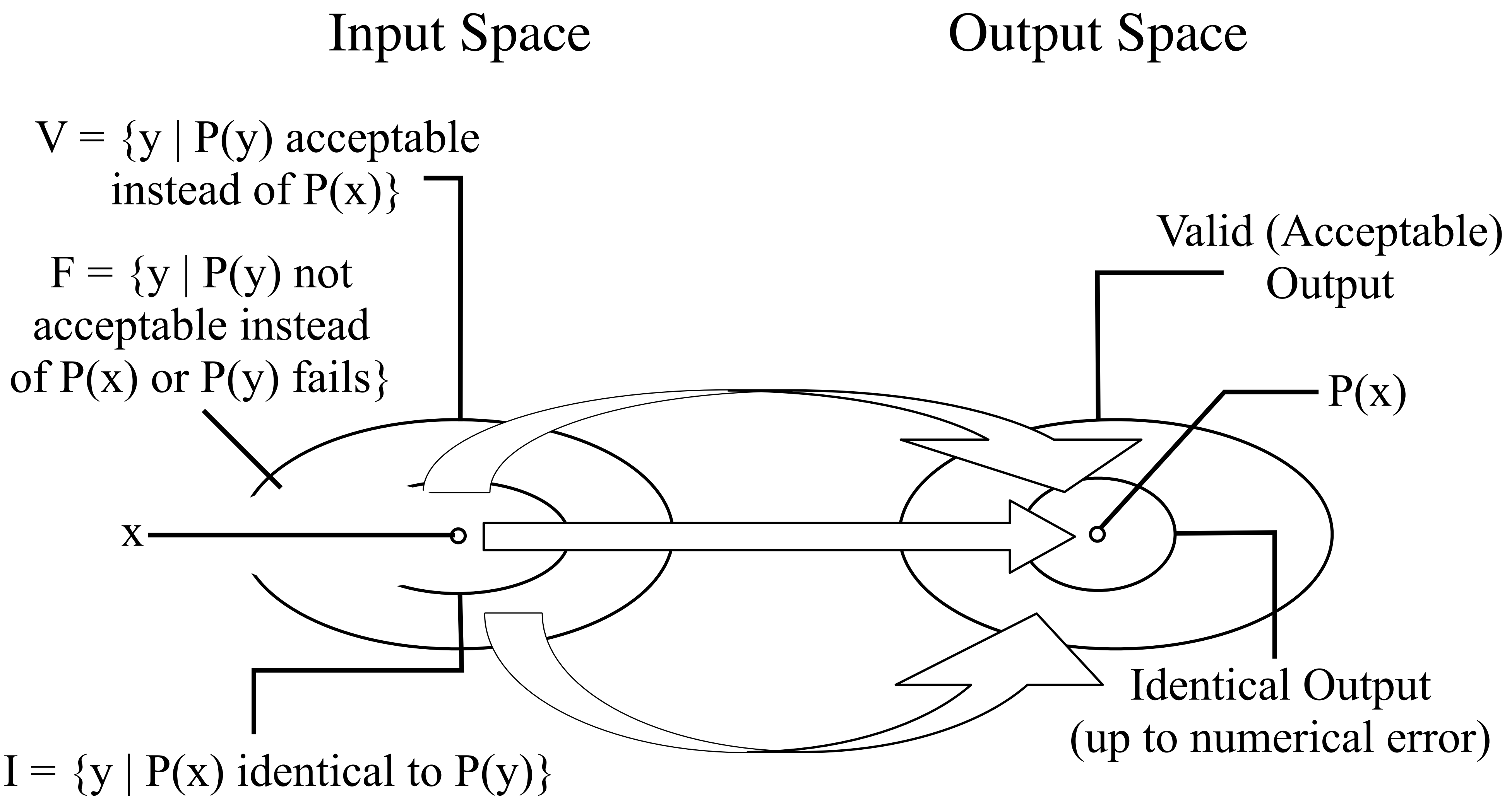
Data Diversity

- Problem with Design Diversity
  - Different alternates need to be developed ⇒ Higher development cost
- Idea of Data Diversity
  - Execute the same software / algorithm with related input, then use a decision algorithm [AK87]
- Based on (application dependent) data re-expression algorithms (DRA)
- The DRA should be simple (fast, and fault-free)
- Complement for design diverse techniques

Data Diversity Definitions (1)

- Input space / output space of a program:
  A hyperspace of many dimensions, defined by the specification
- Failure Domain [Cri89]:
  Set of input points that cause program failure
- Failure Region:
  "Geometry" / distribution of points in the failure domain
  - Observation: failure regions tend to be associated with transitions in the output space

Data Diversity Definitions (2)



Input Space

Output Space

V = {y | P(y) acceptable instead of P(x)}

F = {y | P(y) not acceptable instead of P(x) or P(y) fails}

I = {y | P(x) identical to P(y)}

Valid (Acceptable) Output

P(x)

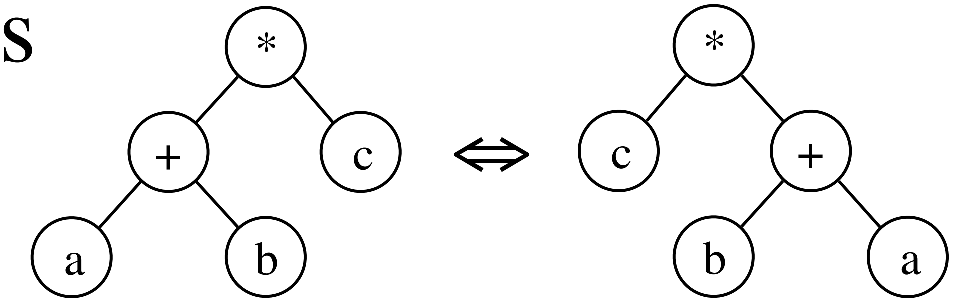
Identical Output (up to numerical error)

Data Re-Expression

- Exact data re-expression algorithms
  - Data re-expression in the set I
  - Transparent outside of the program
  - May unfortunately often preserve the aspect that causes the failure
- Approximate data re-expression algorithms
  - Data re-expression in the set V
  - Better chance of escaping the failure region [AK88]

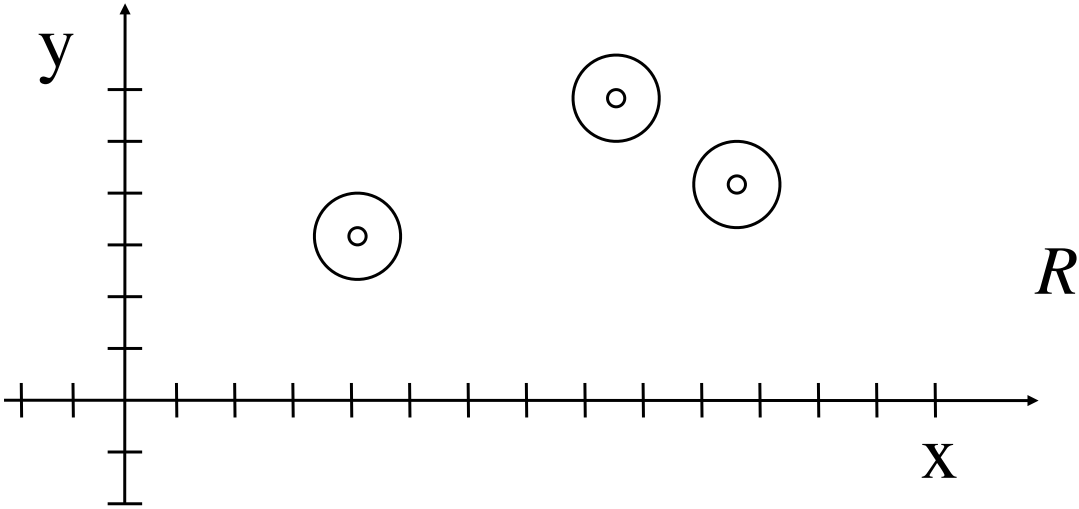
Exact DRA Examples

- Program takes a set of points in a 2D space as an input. Only the relative position of the points is relevant
  - DRA: Translate the coordinate system or rotate the coordinates around an arbitrary point
- Sorting
  - DRA: Random permutation of the input
- Expressions



Approximate DRA Examples

- Introduce low-level “noise” to sensor values
  - Sensors have limited accuracy
  - Perturbing real-world quantities within specific bounds should therefore not affect output



DRA:
Add Small
Random Noise

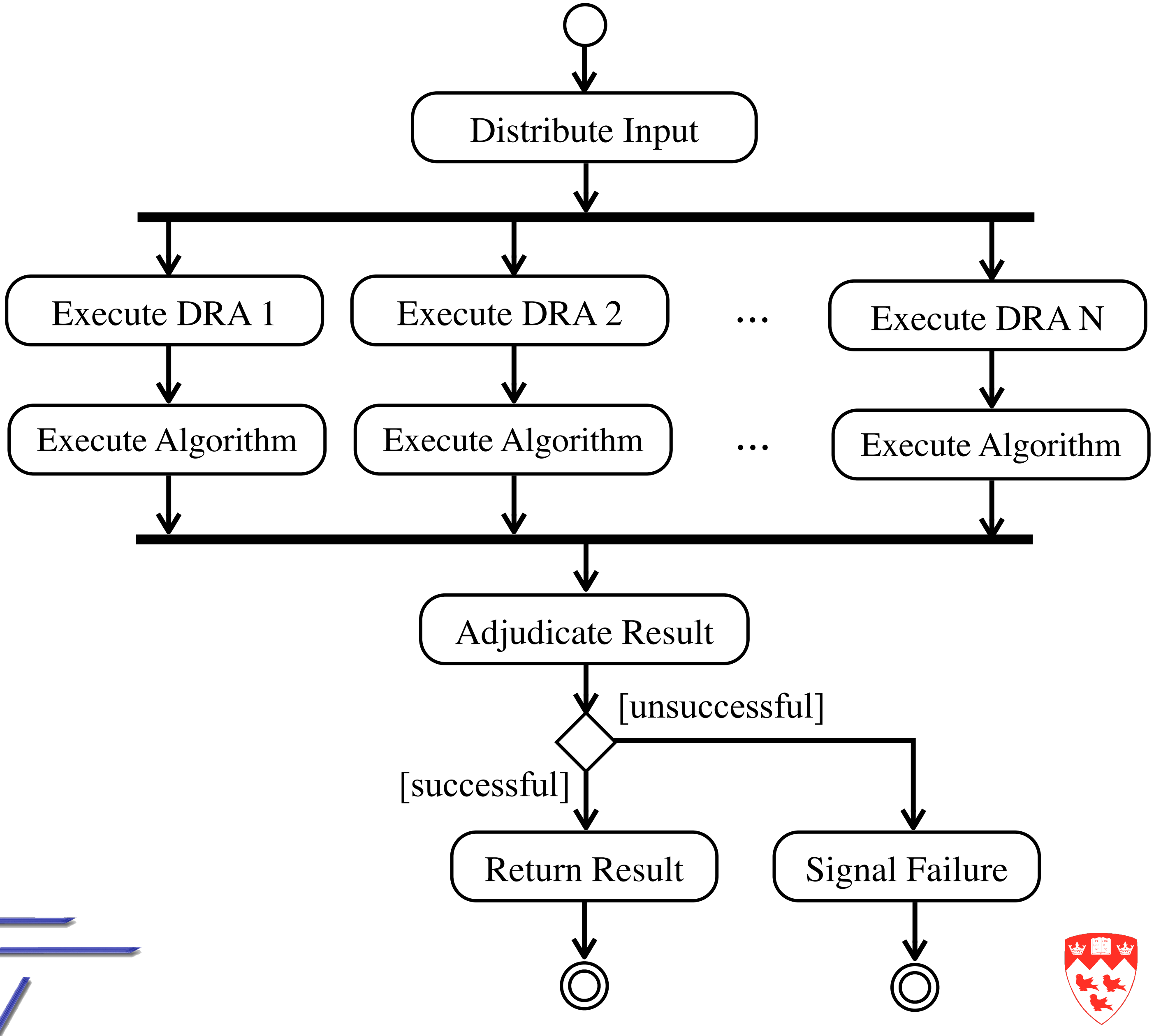
N-Copy Programming (1)

- N (at least 2) versions of the same algorithm run in parallel with slightly different input obtained from the original input and a data re-expression algorithm (DRA)
  - Developed by Ammann and Knight [AK88]
- A decision mechanism selects the "best" result
- Data diverse, static technique

N-Copy Programming (2)

```
run DRA 1 .. DRA n in parallel
run Copy 1 (Result of DRA 1) …
    Copy n (Result of DRA n) in parallel
if Decision Mechanism
  (Result 1, .. Result n) return Result
else signal failure exception
```

N-Copy Programming Execution



N-Copy Programming Discussion

- Runs in a multiprocessor environment
- Small run-time overhead
  - Running the (slowest) data re-expression algorithm
  - Running the decision algorithm
  - Synchronization
- Continuity of service

Design Diversity: Experimental Results

- The major cause of common faults are flawed specifications (incompleteness / ambiguity)
  - Using diverse specifications raises the problem of proving equivalence
- Programmers tend to make similar mistakes
- Coincident failures are less likely if different development processes are used for each variant
- Fewer faults in strongly typed languages

Questions

- What must be part of a specification for a system that is to be designed using design diverse fault tolerance techniques?
- What is the consistent comparison problem?
- What are confident signals?

References (1)

- [Avi85]
  Avizienis, A.: “The N-version Approach to Fault-Tolerant Software”, IEEE Transactions on Software Engineering, Vol. SE-11, No. 12, pp. 1491-1501, 1985.
- [Elm72]
  Elmendorf, W. R.: “Fault Tolerant Programming”, Proceedings of FTCS-2, Newton, MA, pp. 79 - 83, 1972.
- [CA78]
  Chen, L. and Avizienis, A.: “N-Version Programming: A Fault Tolerance Approach to Reliability of Software Operation”, Proceedings of FTCS-8, Toulouse, France, pp. 3- 9, 1978.
- [LA90]
  Lee, P. A.; Anderson, T.: “Fault Tolerance - Principles and Practice”, in Dependable Computing and Fault-Tolerant Systems, Springer Verlag, 2nd ed., 1990.
- [EL85]
  Eckhardt, D. E.; Lee, L. D.: “A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors”, IEEE Transactions on Software Engineering, Vol. SE-11, No. 12, pp. 1511-1517, 1985.

References (1)

- [LM89]
  Littlewood, B.; Miller, D. R.: "Conceptual Modeling of Coincident Failures in Multiversion Software", IEEE Transactions on Software Engineering, Vol. 15, No. 12, pp. 1596-1614, 1980.
- [BKL87]
  Brilliant, S.; Knight, J. C.; Leveson, N. G.: "The Consistent Comparison Problem in N-Version Software", ACM SIGSOFT Software Engineering Notes, Vol 12, No. 1, pp. 29-34, 1987.
- [BS90]
  Blough, D. N. and Sullivan, G. F.: "A Comparison of Voting Strategies for Fault-Tolerant Distributed Systems", Proceedings of the 9th Symposion on Reliable Distributed Systems, Huntsville, AL, USA, pages 136 - 145, 1990.
- [GS90]
  Di Giandomenico, F.; Stringini, L.: "Adjudicators for Diverse-Redundant Components", Proceedings of the 9th Symposion on Reliable Distributed Systems, Huntsville, AL, USA, pages 114 - 123, 1990.

References (1)

- [V93]
  Vouk, M. A. et al.: “An Empirical Evaluation of Consensus Voting and Consensus Recovery Block Reliability in the Presence of Failure Correlation”, Journal of Computer and Software Engineering, Vol. 1, No. 4, pages 367 - 388, 1993.
- [Bis95]
  Bishop, P.: “Software Fault Tolerance by Design Diversity”, in M. R. Lyu (ed.), Software Fault Tolerance, John Wiley & Sons, pp. 211-229, 1995.
- [AK88]
  Ammann, P. E.; Knight, J. C.: “Data Diversity: An Approach to Software Fault Tolerance”, IEEE Transactions on Computers, Vol. 37, No. 4, pp. 418 - 425, 1988.
- [H88]
  G. Hagelin, “ERICSSON Safety Systems for Railway Control”, in Software diversity in computerized control systems (U. Voges, Ed.), 2, pp.11-21, Springer-Verlag, 1988.
- [L90]
  Laprie, J. C. et al.: “Definition and Analysis of Hardware- and Software Fault-Tolerant Architectures”, IEEE Computer, Vol 23, No. 7, pp. 1502 - 1510, 1990.