Faults

1. Major issue in distributed systems is faults (hardware or software)

2. Strategies for handling faults: fault detection and tolerance

3. Fault detection: aim is to detect a fault before it causes serious problems

4. Fault tolerance: proper system operation continues in presence of faults
Faults

1. Major issue in distributed systems is faults (hardware or software)
2. Strategies for handling faults: fault detection and tolerance
1. Major issue in distributed systems is faults (hardware or software)
2. Strategies for handling faults: fault detection and tolerance
3. **Fault detection**: aim is to detect a fault before it causes serious problems
1. Major issue in distributed systems is faults (hardware or software)
2. Strategies for handling faults: fault detection and tolerance
3. Fault detection: aim is to detect a fault before it causes serious problems
4. Fault tolerance: proper system operation continues in presence of faults
Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.

Redundancy of components: basic technique for fault detection and tolerance

Consider replacing one component that on input gives an output by
1. three copies of the system using a splitter and a voter
2. on input the splitter sends it to each duplicated component
3. the voter accepts outputs and outputs majority value

It works in presence of both “transient” and “permanent” faults

Let TMR be triple modular redundancy.
Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by

1. three copies of the system using a splitter and a voter
2. on input the splitter sends it to each duplicated component
3. the voter accepts outputs and outputs majority value

It works in presence of both “transient” and “permanent” faults

Let TMR be triple modular redundancy.
Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.
- **Redundancy of components:** basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component
  3. the voter accepts outputs and outputs majority value
- It works in presence of both “transient” and “permanent” faults
- Let TMR be triple modular redundancy.
Triple modular redundancy

Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.

Redundancy of components: basic technique for fault detection and tolerance

Consider replacing one component that on input gives an output by

1. three copies of the system using a splitter and a voter
2. on input the splitter sends it to each duplicated component

It works in presence of both “transient” and “permanent” faults

Let TMR be triple modular redundancy.
Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.
- Redundancy of components: basic technique for fault detection and tolerance
- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component
  3. the voter accepts outputs and outputs majority value

It works in presence of both “transient” and “permanent” faults

Let TMR be triple modular redundancy.
Triple modular redundancy

- Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.

- **Redundancy of components:** basic technique for fault detection and tolerance

- Consider replacing one component that on input gives an output by
  1. three copies of the system using a splitter and a voter
  2. on input the splitter sends it to each duplicated component
  3. the voter accepts outputs and outputs majority value

- It works in presence of both “transient” and “permanent” faults
Triple modular redundancy

▶ Following is from Bruns, “Distributed systems analysis with CCS”, Prentice-Hall, 1997.
▶ **Redundancy of components**: basic technique for fault detection and tolerance

▶ Consider replacing one component that on input gives an output by

1. three copies of the system using a splitter and a voter
2. on input the splitter sends it to each duplicated component
3. the voter accepts outputs and outputs majority value

▶ It works in presence of both “transient” and “permanent” faults

▶ Let TMR be triple modular redundancy.
Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system.
Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system

- Agent $S$ receives input at $\text{in}$ and passes it to the modules $M_i$, $1 \leq i \leq 3$
Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system

- Agent $S$ receives input at $\text{in}$ and passes it to the modules $M_i$, $1 \leq i \leq 3$

- Agent $M_i$ receives input at port $m_i$; and passes output at $\overline{m_0}$ which may be corrupted
Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system
- Agent $S$ receives input at $\text{in}$ and passes it to the modules $M_i$, $1 \leq i \leq 3$
- Agent $M_i$ receives input at port $m_i$ and passes output at $\overline{m_o}$ which may be corrupted
- Agent $V$ receives outputs at $m_o$ and passes the majority output value to $\text{out}$

Add acknowledgement between $V$ and $S$
Simple TMR

- Describe a simple TMR system and show that if the number of simultaneous faults is at most one then it behaves the same as a fault-free system.

  - Agent $S$ receives input at $\text{in}$ and passes it to the modules $M_i$, $1 \leq i \leq 3$.

  - Agent $M_i$ receives input at port $m_i$ and passes output at $\overline{m_o}$ which may be corrupted.

  - Agent $V$ receives outputs at $m_o$ and passes the majority output value to $\text{out}$.

  - Add acknowledgement between $V$ and $S$. 

Simple TMR II

\[ S \overset{\text{def}}{=} \text{in}(x). (\overline{\text{mi}}_1(x). (\overline{\text{mi}}_2(x). \overline{\text{mi}}_3(x). S' + \overline{\text{mi}}_3(x). \overline{\text{mi}}_2. S')) \\
+ (\overline{\text{mi}}_2(x) \ldots ) \\
+ (\overline{\text{mi}}_3(x) \ldots ) \ldots . . . . . ) \]

\[ S' \overset{\text{def}}{=} \text{ok}. S \]

\[ M_i \overset{\text{def}}{=} \text{mi}_i(x). (\overline{\text{mo}}(x). M_i + \sum \{ \overline{\text{mo}}(v). M_i : v \in D \}) \]

\[ V \overset{\text{def}}{=} \text{mo}(x_1). \text{mo}(x_2). \text{mo}(x_3). \\
\text{if } x_1 = x_2 \text{ then out}(x_1). V' \text{ else out}(x_3). V' \]

\[ V' \overset{\text{def}}{=} \overline{\text{ok}}. V \]

\[ \text{TMR}_1 \equiv (S|M_1|M_2|M_3|V) \backslash \{ \text{mi}_i, \text{mo}, \text{ok} \} \]
Simple TMR II

\[ S \overset{\text{def}}{=} \text{in}(x). (\overline{m_i_1}(x). \overline{m_i_2}(x). \overline{m_i_3}(x). S' + \overline{m_i_3}(x). \overline{m_i_2}. S') + (\overline{m_i_2}(x) \ldots) + (\overline{m_i_3}(x) \ldots) \]

\[ S' \overset{\text{def}}{=} \text{ok}. S \]

\[ M_i \overset{\text{def}}{=} \text{mi}_i(x). (\overline{m_o}(x). M_i + \sum \{ \overline{m_o}(v). M_i : v \in D \}) \]

\[ V \overset{\text{def}}{=} \text{mo}(x_1). \text{mo}(x_2). \text{mo}(x_3). \]  

\[ \text{if } x_1 = x_2 \text{ then } \overline{\text{out}}(x_1). V' \text{ else } \overline{\text{out}}(x_3). V' \]

\[ V' \overset{\text{def}}{=} \overline{\text{ok}}. V \]

\[ \text{TMR}_1 \equiv (S | M_1 | M_2 | M_3 | V) \setminus \{ \text{mi}_i, \text{mo}, \text{ok} \} \]

\text{Note TMR}_1 \not\approx \text{Cop} \quad \text{Why?}
Need to capture that \( TMR_1 \) behaves like \( \text{Cop} \) if at most one module produces a fault.
Simple TMR III

- Need to capture that $\text{TMR}_1$ behaves like $\text{Cop}$ if at most one module produces a fault.
- **Exercise:** How to do this?
Simple TMR III

- Need to capture that $\text{TMR}_1$ behaves like $\text{Cop}$ if at most one module produces a fault.
- Exercise: How to do this?
- Let $MP_i$, $1 \leq i \leq 3$, be a perfect module.

\[ MP_i \overset{\text{def}}{=} \text{mi}_i(x).\overline{\text{mo}}(x).MP_i \]
Simple TMR III

- Need to capture that $TMR_1$ behaves like $Cop$ if at most one module produces a fault.

- Exercise: How to do this?

- Let $MP_i$, $1 \leq i \leq 3$, be a perfect module.

  $$MP_i \overset{\text{def}}{=} mi_i(x).\overline{mo}(x).MP_i$$

- Instead of

  $$TMR_1 \equiv (S|M_1|M_2|M_3|V)\{mi_i, mo, ok\}$$

  assume just one faulty module

  $$TMR'_1 \equiv (S|M_1|MP_2|MP_3|V)\{mi_i, mo, ok\}$$
Simple TMR III

Need to capture that $TMR_1$ behaves like $Cop$ if at most one module produces a fault.

Exercise: How to do this?

Let $MP_i$, $1 \leq i \leq 3$, be a perfect module.

$$MP_i \overset{\text{def}}{=} \text{mi}_i(x).\text{mo}(x).MP_i$$

Instead of

$$TMR_1 \equiv (S|M_1|M_2|M_3|V)\{\text{mi}_i,\text{mo},\text{ok}\}$$

assume just one faulty module

$$TMR'_1 \equiv (S|M_1|MP_2|MP_3|V)\{\text{mi}_i,\text{mo},\text{ok}\}$$

Now $TMR'_1 \approx Cop$
Simple TMR III

Need to capture that $\text{TMR}_1$ behaves like $\text{Cop}$ if at most one module produces a fault.

Exercise: How to do this?

Let $MP_i$, $1 \leq i \leq 3$, be a perfect module.

$$MP_i \overset{\text{def}}{=} mi_i(x).\overline{mo}(x).MP_i$$

Instead of

$$\text{TMR}_1 \equiv (S|\text{M}_1|\text{M}_2|\text{M}_3|V)\\{\text{mi}_i,\text{mo},\text{ok}\}$$

assume just one faulty module

$$\text{TMR}_1' \equiv (S|\text{M}_1|MP_2|MP_3|V)\\{\text{mi}_i,\text{mo},\text{ok}\}$$

Now $\text{TMR}_1' \approx \text{Cop}$

Exercise: produce the weak bisimulation
A more realistic TMR involves error detection.

- the interface includes fault; and detect; ports (as well as in and out)

- fault models module faults

- to detect faults we add to each basic module a disagreement detector that compares the value computed by the module with the majority value reported by voter.

- Components S, splitter M, and D, modules and detectors V, voter
A more realistic TMR involves error detection.

- the interface includes fault\textsubscript{i} and detect\textsubscript{i} ports (as well as in and out)
- fault\textsubscript{i} models module faults
A more realistic TMR involves error detection.

- the interface includes fault; and detect; ports (as well as in and out)
- fault; models module faults
- to detect faults we add to each basic module a disagreement detector that compares the value computed by the module with the majority value reported by voter.
A more realistic TMR involves error detection.

- the interface includes fault; and detect; ports (as well as in and out)
- fault; models module faults
- to detect faults we add to each basic module a disagreement detector that compares the value computed by the module with the majority value reported by voter.

**Components**
- $S$ splitter
- $M_i$ and $D_i$ modules and detectors
- $V$ voter
TMR with error detection II

\[
S \overset{\text{def}}{=} \text{in}(x). (\overline{\text{mi}}_1(x). (\overline{\text{mi}}_2(x). \overline{\text{mi}}_3(x). \text{ok}. S + \overline{\text{mi}}_3(x). \overline{\text{mi}}_2. \text{ok}. S) \\
+ (\overline{\text{mi}}_2(x) \ldots) + (\overline{\text{mi}}_3(x) \ldots) \ldots)
\]

\[
M'_i \overset{\text{def}}{=} \text{mi}_i(x). (\overline{\text{mo}_i}(x). M'_i + \text{fault}. \sum \{\overline{\text{mo}_i}(v). M'_i : v \in D\})
\]

\[
D_i \overset{\text{def}}{=} \text{mo}_i(x). \text{do}(x). D'_i(x)
\]

\[
D'_i(x) \overset{\text{def}}{=} \text{vo}(y). (\text{if } x \neq y \text{ then } \text{detect}_{i}. D_i \text{ else } D_i)
\]

\[
V' \overset{\text{def}}{=} \text{do}(x_1). \text{do}(x_2). \text{do}(x_3). \text{if } x_1 = x_2 \text{ then } V''(x_1) \text{ else } V''(x_3)
\]

\[
V''(x) \overset{\text{def}}{=} \overline{\text{vo}}(x). \overline{\text{vo}}(x). \overline{\text{vo}}(x). \text{out}(x). \text{ok}. V'
\]

\[
\text{TMR}_2 \equiv (S | M'_1 | D_1 | M'_2 | D_2 | M'_3 | D_3 | V') \setminus \{\text{mi}_i, \text{do}_i, \text{vo}_i, \text{mo}_i, \text{ok}\}
\]
What is the relationship between TMR$_1$ and TMR$_2$?
What is the relationship between \( \text{TMR}_1 \) and \( \text{TMR}_2 \)?

Problem \( \text{TMR}_2 \) has observable actions \underline{fault} and \underline{detect}; (besides \underline{in} and \underline{out})
What is the relationship between TMR\(_1\) and TMR\(_2\)?

Problem TMR\(_2\) has observable actions fault and detect; (besides in and out)

How can we “abstract” from them?
Abstracting actions I

- Suppose we have system $W$ that can do actions $K$ and system $W'$ that can do $K$ and the extra action $a$.
Abstracting actions I

- Suppose we have system $W$ that can do actions $K$ and system $W'$ that can do $K$ and the extra action $a$.
- We want to relate $W$ and $W'$. We can abstract from $a$ by “transforming” it into $\tau$.

Let $A \overset{\text{def}}{=} a$.

Let $W'' \equiv (W' | A) \{ a \}$.

Now we can ask: $W \approx W''$?
Abstracting actions I

- Suppose we have system $W$ that can do actions $K$ and system $W'$ that can do $K$ and the extra action $a$.
- We want to relate $W$ and $W'$. We can abstract from $a$ by “transforming” it into $\tau$.
- Let $A \overset{\text{def}}{=} \overline{a}.A$
Abstracting actions I

- Suppose we have system $W$ that can do actions $K$ and system $W'$ that can do $K$ and the extra action $a$.
- We want to relate $W$ and $W'$. We can abstract from $a$ by "transforming" it into $\tau$.
- Let $A \overset{\text{def}}{=} \overline{a}.A$
- Let $W'' \equiv (W'|A)\{a\}$
Suppose we have system $W$ that can do actions $K$ and system $W'$ that can do $K$ and the extra action $a$.

We want to relate $W$ and $W'$. We can abstract from $a$ by “transforming” it into $\tau$.

Let $A \overset{\text{def}}{=} \overline{a}.A$

Let $W'' \equiv (W'|A)\backslash\{a\}$

Now we can ask: $W \approx W''$?
Abstracting actions II

Abstract from actions

$$\text{Ab} \overset{\text{def}}{=} \text{fault}.\text{Ab} + \sum \{\text{detect}_i.\text{Ab} : 1 \leq i \leq 3\}$$

$$\text{TMR}_2' \equiv (\text{TMR}_2|\text{Ab})\setminus \{\text{fault}, \text{detect}\}$$
Abstracting actions II

- **Abstract from actions**

  \[
  \text{Ab} \overset{\text{def}}{=} \text{fault}.\text{Ab} + \sum_{i=1}^{3} \{\text{detect}_i.\text{Ab} : 1 \leq i \leq 3\}
  \]

  \[
  \text{TMR}_2' \equiv (\text{TMR}_2|\text{Ab})\backslash\{\text{fault}, \text{detect}_i\}
  \]

- **Exercise** Prove that \(\text{TMR}_1 \approx \text{TMR}_2'\)
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services

- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...

- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)

- Correctness through equivalence and model-checking

- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems

- Requires changes to basic model of transition graphs

- Correctness is more complex (timed/probabilistic/... bisimulations and temporal logics)

- Finish course: algorithms for model checking and equivalence checking on finite transition systems
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems

Finish course: algorithms for model checking and equivalence checking on finite transition systems
Lots more examples of systems defined in CCS: recent example is web services

Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...

Maintain basic model of transition systems (vertices as states/processes, edges as transitions)

Correctness through equivalence and model-checking

Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems

Requires changes to basic model of transition graphs
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ... 
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems
- Requires changes to basic model of transition graphs
- Correctness is more complex (timed/probabilistic/... bisimulations and temporal logics)
Concurrent systems: alternatives/extensions

- Lots more examples of systems defined in CCS: recent example is web services
- Alternatives other process calculi (CSP, ...), petri nets, IO-automata, ...
- Maintain basic model of transition systems (vertices as states/processes, edges as transitions)
- Correctness through equivalence and model-checking
- Extensions pi-calculus (for mobility), adding quantities (time, probability, ...) for modelling embedded/hybrid/biological systems
- Requires changes to basic model of transition graphs
- Correctness is more complex (timed/probabilistic/... bisimulations and temporal logics)
- Finish course: algorithms for model checking and equivalence checking on finite transition systems