Interaction Testing

Chapter 15

Interaction faults and failures

- Subtle
  - **Difficult to detect with testing**
  - **Usually seen after systems have been delivered**
    - **In low probability threads**
    - **Occur after a long time**
      - Large numbers of thread executions
    - **Difficult to reproduce**

Interaction faults and failures – 2

- To be able to test interactions need
  - **To understand what they are**
  - **Mathematical description**
    - **Look at requirements specification**
- Concerned with unexpected interactions

Context of interaction

- It is a relationship **InteractsWith** among
  - Data
  - Events
  - Threads
  - Actions
  - Ports
  - The relationship is reflexive
  - It is binary relation between
    - Data & events
    - Data & threads
    - Events & threads

Context of interaction– 2

- There are too many relationships to be of direct use
  - **Indicates that something is missing**
  - **In this case location**
    - **Place and time**

- Select location to be an attribute of the other entities instead of being a new entity
  - **Short coming of requirements to not include it**

Meaning of the location attribute

- Place
  - **Have a coordinate system**
  - **For software use processor residence**
    - **Only useful for multi-processor systems**

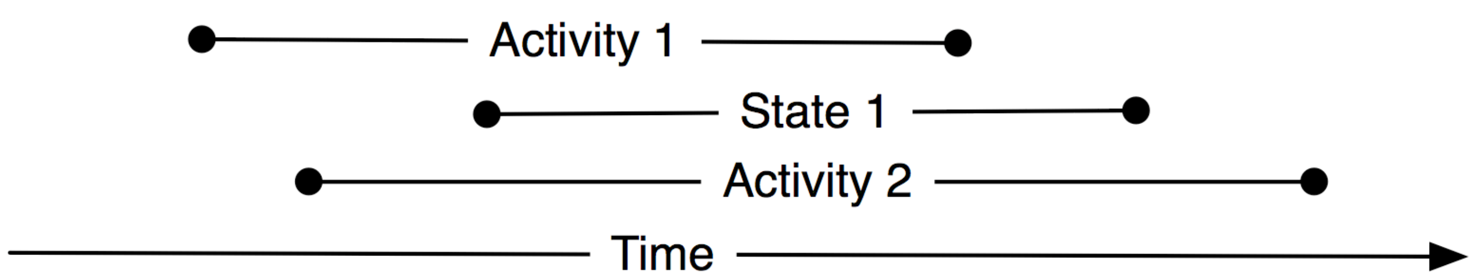
Meaning of the location attribute – 2

- Time
  - **An instant**
    - **When something happens**
    - **Ask before and after type questions**
  - **An interval**
    - **Interested in duration**

Events & states

- Textbook has two meanings for event
  - **Causes confusion, ambiguity, wordy explanations**

- Use two words
  - **Use event for instant**
  - **Use state or activity for duration**
    - **Occurs between two events**



Properties of threads and processors

- Threads have duration
  - **They are activities**

- At one time a processor can execute only one thread

Properties of threads and processors – 2

- A processor is in a state of executing a thread
  - **Timesharing, multiprocessing interleaves thread execution**
    - **Processor changes state for each thread**
  - **Here thread durations overlap in time**

Properties of threads and processors – 3

- On one processor events can be simultaneous within the minimum resolution of **time-grain markers**
  - **BUT reality (hardware) puts an order on those events – puts them in a sequence**
    - **As far as we can tell it is a random choice**
    - **At another occurrence the events may be ordered in a different sequence**
      - That is an essential difficulty of interaction testing

Properties of threads and processors – 4

- On different processors, events can occur simultaneously
  - **Common events by definition must occur at the same time**
    - **Consider a two people colliding – the collision is a common event to the two people (processors)**
  - **Synchronous communication for processors start and end with common events**

Properties of threads and processors – 5

- For a single processor
  - **Input and output events occur during thread execution**
  - **From the perspective of a thread they cannot occur simultaneously, because they occur at instructions and instructions are executed sequentially**
  - **From the perspective of devices port events can be simultaneous**
    - **For each port events occur in time sequence**

Properties of threads and processors – 6

- Threads occur only within one processor
  - **Do not cross processor boundaries**
  - **Have trans-processor quiescence when threads reach processor boundaries**
    - **Analogous to crossing unit boundaries in integration testing**

Properties of threads and processors – 7

- What we want is **sane** behaviour
  - **This results from considering events to be in a linear sequence**
    - **For example synchronous communications takes into account message transmission time**
    - **Break the communication into events such as**
      - Sender starts sending
      - Receiver starts receiving
      - Sender ends sending
      - Receiver ends receiving

Properties of threads and processors – 8

- **For interaction faults and failures need to go down to this level**
  - **Implies time-grain markers need to have very fine resolution**

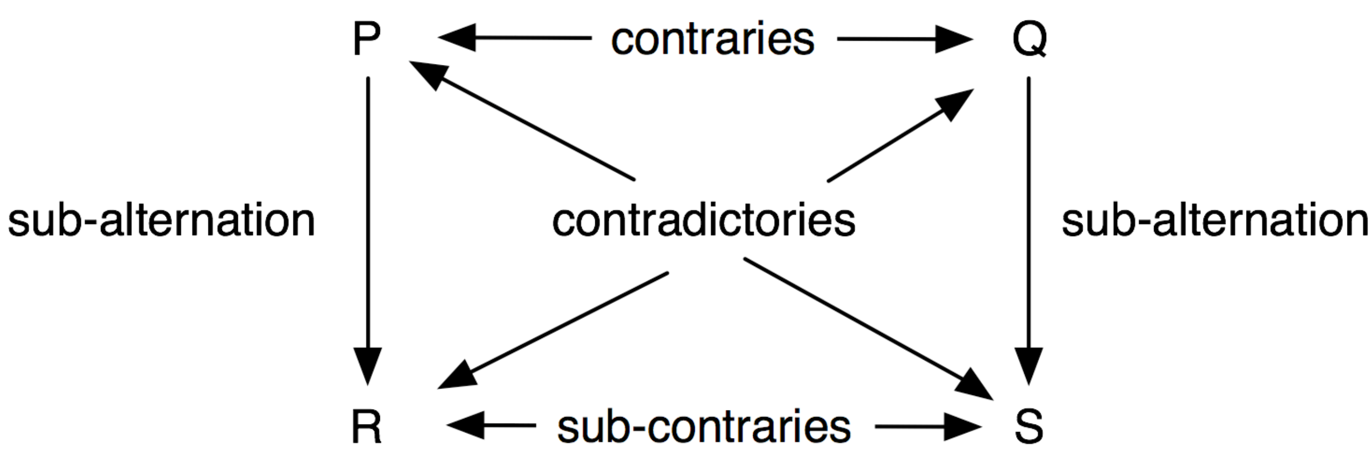
Taxonomy of interactions

- Static interactions in a single processor system
- Static interactions in multiprocessor system
- Dynamic interactions in a single processor system
- Dynamic interactions in multiprocessor system

Processors	Static	Dynamic
Single	Type 1	Type 3
Multiple	Type 2	Type 4

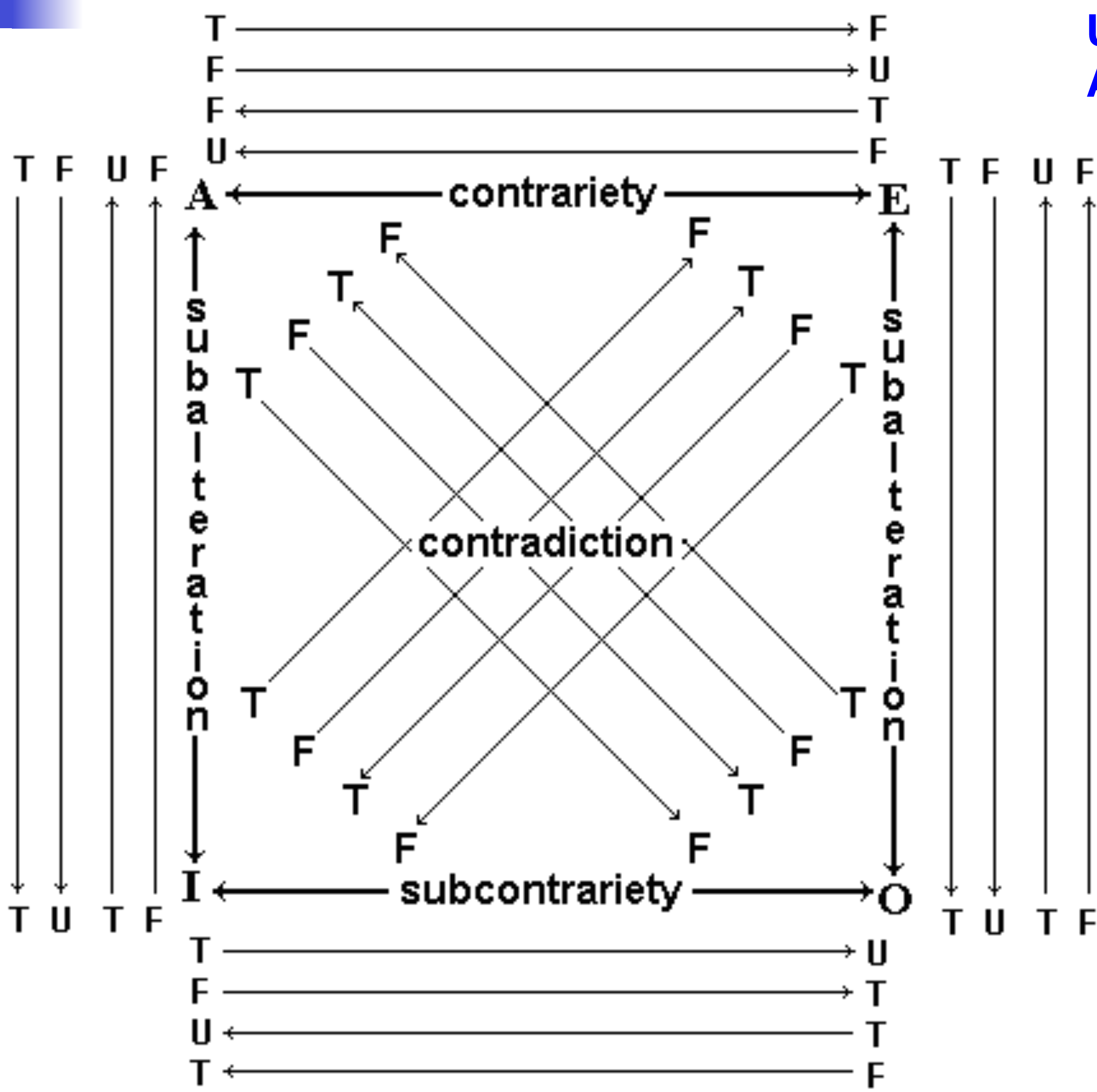
Square of opposition

- Given two propositions P and Q
  - **They are contraries if both cannot be true**
  - **Sub-contraries if both cannot be false**
  - **Contradictories if exactly one is true**
  - **R is a subaltern of P if the truth of P guarantees the truth of R – i.e. P → R**



Square of opposition – 2

U – undetermined
Arrow is implication



Why logic?

- Consider the following data interactions
  - **Precondition for a thread is a conjunction of data propositions**
    - **Contrary or contradictory data values prevent execution**
  - **Context-sensitive input port events can involve contradictory or contrary data**
  - **Case statement clauses, if correct, are contradictories**
  - **Rules in a decision table, if correct, are contradictories**

Static interactions in a single processor

- Analogous to combinatorial circuits
  - **Model with decision tables and unmarked event-driven Petri nets**
  - **Telephone system example**
    - **Call display and unlisted numbers are contraries**
      - Both cannot be satisfied
      - Both could be waived

Static interactions in a multiprocessor

- Location of data is important

Telephone example 1

- Calling party in location of one processor (area)
- Receiving party in another processor
- Checking for contrary data such as caller id and unlisted numbers
  - **Can only check when caller and receiver are connected by a thread**
  - **A contrary relationship exists as a static interaction across multiple processors**
  - **Failure occurs only when the two threads interact**

Telephone example 2

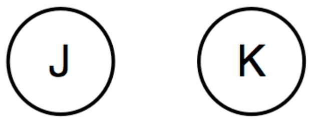
- Call forwarding is defined
  - Alice (area 1) has call forwarding to Bob
  - Bob (area 2) has call forwarding to Charlene
  - Charlene(area 3) has call forwarding to Alice
- The call forwarding data is contrary – cannot all be true at the same time
  - Have distributed contraries
- Call forwarding is a property of a local office
- A thread sets a forwarding location
- Have a fault but not a failure until Donald places a call to one of Alice, Bob or Charlene

Static interactions summary

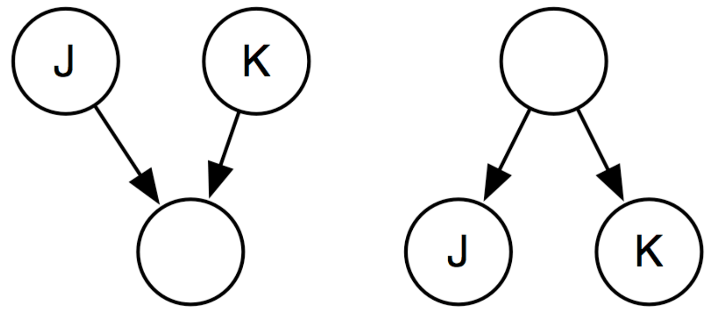
- The same in both single processor and multiprocessor systems
- More difficult to detect in multiprocessor systems
- Functional dependencies in a database (centralized or distributed) are static interactions
  - **Both are a form of subalternation**

Graph connectedness for dynamic interactions

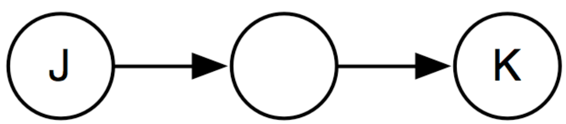
- Make use of n-connectedness in graphs



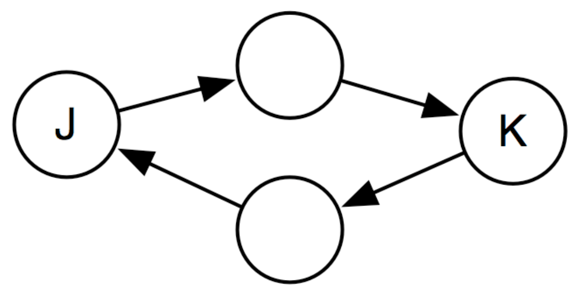
0-connected



1-connected



2-connected



3-connected

Data-data connectedness – Logical relationships

- 0-connected
  - **Logically independent**
- 2-connected
  - **Sub-alternation**
- 3-connected – bidirectional
  - **Contraries**
  - **Contradictories**
  - **Sub-contraries**

Examples

- 1-connected data-data
  - Two or more data items are input to the same action
- 2-connected data-data
  - When a data item is used in a computation

Examples – 2

- 3-connected data-data
  - When data are deeply related, as in repetition and semaphores
- 1-connected data-event
  - Context-sensitive port input events

Dynamic, single processor interactions

- Six potential interaction pairs
  - **Combination pairs of**
    - **Data**
    - **Events**
    - **Threads**

- Each interaction can exhibit 4 different graph connectedness attributes

- Result is 24 sub-categories for these interactions

Dynamic, single processor interactions – 2

- Do not analyze all possibilities
  - **Interaction faults only result in failure when threads establish a connection**
- Thread-thread interaction occurs
  - **Through events**
  - **Through data**

Petri net external inputs and outputs

- External inputs
  - **Places with in-degree 0**
    - **Can be port or data pre-condition place**

- External outputs
  - **Places with out-degree 0**
    - **Can be port or data post-condition place**

For an example
see Figure 15.5

Thread-thread interaction

- Each thread can be represented by an EDPN

- The symbolic names of the places and transitions correspond to those in the EDPN for the system

  - **Synonyms in thread nets need to be resolved when they interact**

Thread-thread interaction – 2

- Threads only interact through external input and output events
  - **The intersection of the external input and output places for the threads indicates where they interact with each other**

For an example
see Figures 15.6 & 15.7

Thread-thread interaction – 3

- External events always remain external
- External data may become internal
  - **Output of one thread is input to another**
    - **Call forwarding**

Thread-thread connectedness definition

- T1 and T2 are threads where EI1, EI2, EO1 and EO2 are the external inputs and outputs of the threads
  - **0-connected**
    - $EI1 \cap EI2 = \emptyset \quad \wedge \quad EO1 \cap EO2 = \emptyset$
      $EO2 \cap EI1 = \emptyset \quad \wedge \quad EO1 \cap EI2 = \emptyset$
  - **1-connected**
    - $EI1 \cap EI2 \neq \emptyset \quad \oplus \quad EO1 \cap EO2 \neq \emptyset$
  - **2-connected – only through data places**
    - $EO1 \cap EI2 \neq \emptyset \quad \oplus \quad EI1 \cap EO2 \neq \emptyset$
  - **3-connected – only through data places**
    - $EO1 \cap EI2 \neq \emptyset \quad \wedge \quad EI1 \cap EO2 \neq \emptyset$

Directed thread graph

- A directed thread graph can be constructed
  - **Nodes are threads**
    - **External inputs & outputs are not in the node**
      - They remain external to the node.
    - **Edges connect threads according to the external common input & output ports, and common data places**
      - Figure 15.8 is an example made from Figure 15.7
- Can see connectedness relationships

1-connected threads

- 1-connected threads from input places are the typical case for Petri-net mutual exclusion
  - **A token on the common input is consumed by one of the threads and other cannot proceed**

- 1-connected threads to output places have an ambiguity
  - **We do not know which thread produced an output token**
    - **Can occur from unexpected thread interaction where some threads completed execution earlier**

For an example
see Figure 15.7

2- and 3-connected threads

- Can only occur with data places
  - **Port places cannot be both input an output**
    - **Note some devices may have both input and output capability but we always split into independent input and output logical devices**

2- and 3-connected threads – 2

- Problem is often time difference between the setting of data and the occurrence of a failure due to thread interaction
  - **Read-only data has infinite duration**
    - **Rarely causes problems**
  - **Read / write data has a duration**
    - **Problem is caused by an earlier write that has been replaced**
      - Can be very difficult to diagnose and test

Thread interaction Warning

Problems occur when we

Expect 0-connectedness

But have 1-, 2- or 3-connectedness

Dynamic, multi-processor interactions

- Problem here is threads and events occur in parallel
  - **We have concurrent behaviour with a collection of communicating sequential processors (CSP)**
  - **Have non-deterministic behaviour**
  - **To fully understand need to learn the mathematics of CSP**
    - **Without that can only work through an example**
      - Figures and tables in Section 15.2.4

Dynamic, multi-processor interactions – 2

- Difficulties arise from
  - **Combined Petri nets grow exponentially in size and complexity**
  - **May be difficult to rationalize initial marking**
  - **Have mutual exclusion**
    - **Contraries**

Dynamic, multi-processor interactions – 3

- Difficulties arise from
  - **What is the duration of an output**
    - **Is it controlled by the Petri net?**
    - **Or fixed in some way?**
  - **Time interval between events and model reaction time**
    - **What happens to data values**
    - **Output events**
  - **Have non-deterministic systems**

Deterministic system

- **How can you tell if a system is deterministic?**

Informal definition of determinism

- (1) A system is deterministic if, given its inputs, we can always predict its outputs

- (2) A system is deterministic if it always produces the same outputs for a given set of inputs

Informal definition of determinism – 2

- (For a non-deterministic system it may be difficult to demonstrate different output
    - **Process P chooses non-deterministically at every step whether to engage in event 'a' or 'b'**
    - **Process Q chooses non-deterministically once whether to engage only with event 'a' or only with event 'b'**

$$\textit{traces}(\mathrm{Q}) \subset \textit{traces}(P)$$

$P = (a \rightarrow P) \sqcap (b \rightarrow P)$

$Q = (a \rightarrow Qa) \sqcap (b \rightarrow Qb)$

$Qa = (a \rightarrow Qa)$

$Qb = (b \rightarrow Qb)$

Formal definition of determinism

- P is deterministic $\leftrightarrow \forall s : \mathit{traces}(P) \bullet$

$$X \in \mathit{refusals}(P / s) \leftrightarrow X \cap (P / s)^1 = \{\}$$

$$P^1 = \{ e \mid \langle e \rangle \in \mathit{traces}(P) \}$$

  - **A system is deterministic if at every step the system never refuses to engage in any external event appropriate at that step**

Formal definition of determinism – 2

- P is deterministic $\leftrightarrow \forall s : \mathit{traces}\,(P) \bullet$
  $X \in \mathit{refusals}\,(P / s) \leftrightarrow X \cap (P / s)^1 = \{\}$
  $P^1 = \{ e \mid \langle e \rangle \in \mathit{traces}\,(P) \}$

  - **P^1 definition is the set of events in which P may engage on the first step**
  - **P / s is the process after P has engaged in all of the events in the trace s**
  - **A trace is a record of the external events in which a process has engaged**
  - **A refusal is a set of events in which a process refuses to engage**

On non-determinism

- In a Petri net non-determinism arises when two or more transitions are enabled
  - **Which transition fires is random**
  - **The choice can be made by**
    - **An external event**
      - Environment chooses
    - **An internal event**
      - System chooses
      - Not stated in the textbook

On non-determinism – 2

- **Deadlock** occurs when no transition fires
  - **Bad but at least detectable**
- **Livelock** occurs when internal events take over
  - **Even if an external event is available, the system chooses an internal event**
    - **Basis of infinite loops in programs**
    - **What can happen when a program does not respond to keyboard or mouse**

On non-determinism – 3

- A thread is locally non-deterministic if we cannot predict its output with information local to the thread
  - **In many cases non-determinism vanishes when sufficient context is provided**
    - **Changing the lever in windshield wiper cannot determine output**
    - **By adding in the dial, the output can be determined**

On non-determinism – 4

- Implication for testers
  - **When testing threads with external inputs – especially data – it is necessary to test the interaction with all other threads that can be n-connected (n > 0) via external inputs**

Models and interactions

- Static interactions
  - **Decision tables are models of choice**
- Dynamic interactions – 1 processor
  - **Finite state machines are models of choice**
- Dynamic interactions – multiple processor
  - **Event-driven Petri nets are models of choice**

Client / Server complexities

- Base system has program components
  - **Database, application, presentation (logical output)**
  - **Have a centralized, fat server and fat client distinction**
    - **Figure 15.13**
- Entire system includes above items plus
  - **Network**
  - **GUI**
  - **May have homogeneous or heterogeneous processors**

Implication of Client / Server complexity

- When things go wrong
  - **Lots of possibilities for finger pointing take place**

Client / Server testing

- Extend notion of threads beyond an EDPN
    - **CS transaction**
        - **A sequence of threads across EDPN boundaries**
            - Client processor --> network --> application --> DBMS and back again

Client / Server testing – 2

- Much of the system is stable – e.g. DBMS, existing application
  - **Should testing be needed**
    - **Use functional testing – no source text**

Client / Server GUI testing

- Consists of multiple windows that need to be synchronized
  - **Communicating sequential processors (Petri nets)**
- All events are port events
- Have dynamic interactions across multiple processors

Client / Server GUI testing – 2

- Use operational profiles
- Test individual threads
- Then test thread interaction
  - **Big problem if there are multiple clients such as shared bank accounts**