# 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 execution**
    - **Difficult to reproduce**
- 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

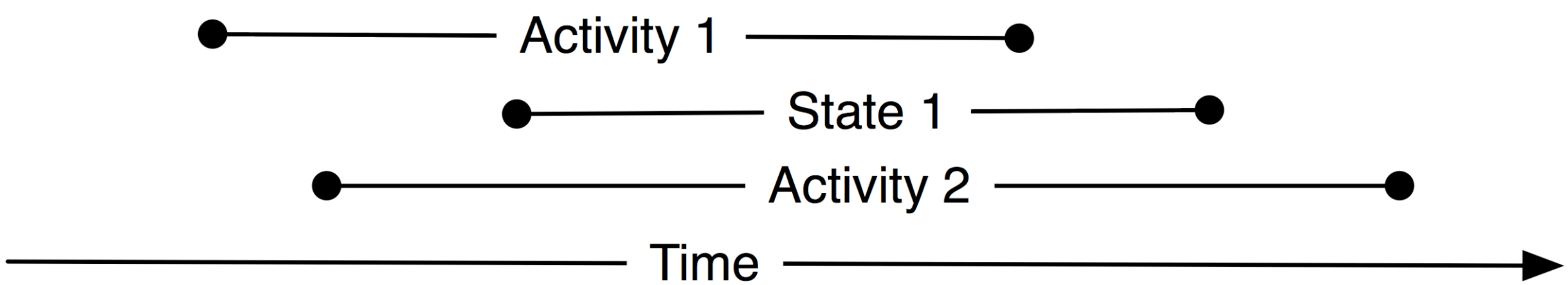
- It is a relationship **InteractsWith** among
  - **Data**
  - **Events**
  - **Threads**
  - **Actions**
  - **Ports**
    - **The relationship reflexive**
    - **It is binary relation between**
      - Data & events
      - Data & threads
      - Events & threads
  - **There are too many relationships to be of direct use**
    - **Indicates that something is missing**
    - **In this case location**
      - Time and place
  - **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

- Time
  - **An instant**
    - **When something happens**
    - **Ask before and after type questions**
  - **An interval**
    - **Interested in duration**
- Location
  - **Have a coordinate system**
  - **For software use processor residence**
    - **What does this mean?  Location is binary – in / out?**

# Events & states

- Two meanings for event
  - **Causes confusion, ambiguity, wordy explanations**
- Use two words
  - **Use event for instant**
  - **Use state, activity for duration**
    - **Occurs between two events**



# Properties of threads and processors

- Threads have duration
  - **They are activities**
- A processor can be executing only one thread at a time
  - **The 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 – 2

- 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 a difficulty in testing interaction
- 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 – 3

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

# Properties of threads and processors – 4

- What we want is **sane** behaviour
  - **This results from considering events to be in a linear sequence**
    - **For example synchronous communications take into account message transmission time – break the communication into events such as**
      - Sender starts sending
      - Receiver receives starts receiving
      - Sender ends sending
      - Receiver ends receiving
  - **For interaction faults and failures need to go down to this level**
    - **Implies time-grain markers need to have very fine resolution**

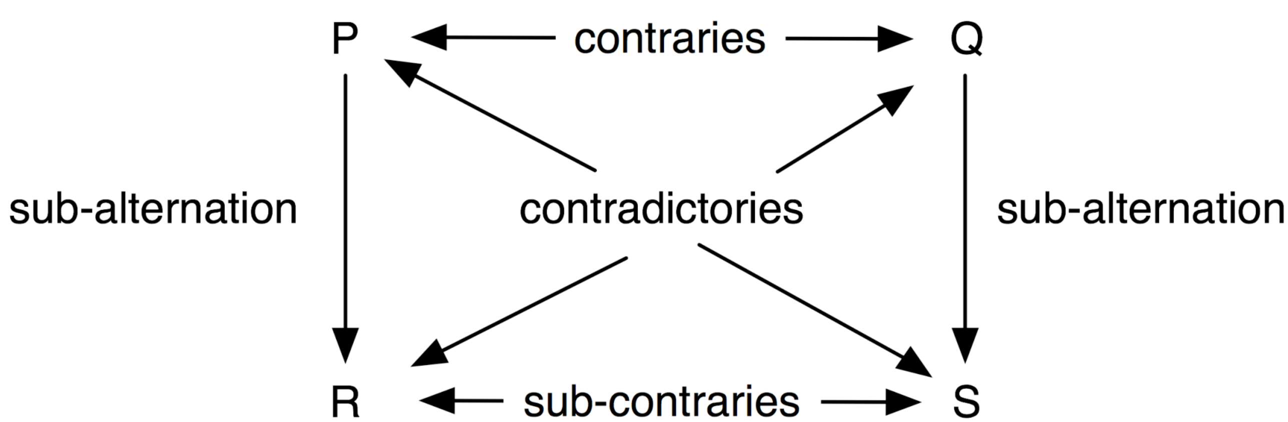
# Taxonomy of interactions

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

| Procesors | 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**
  - **Q is a subaltern of P if the truth of P guarantees the truth of Q – i.e. P → Q**



# 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 usually 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 when they interact**

# Static interactions in a multiprocessor – 2

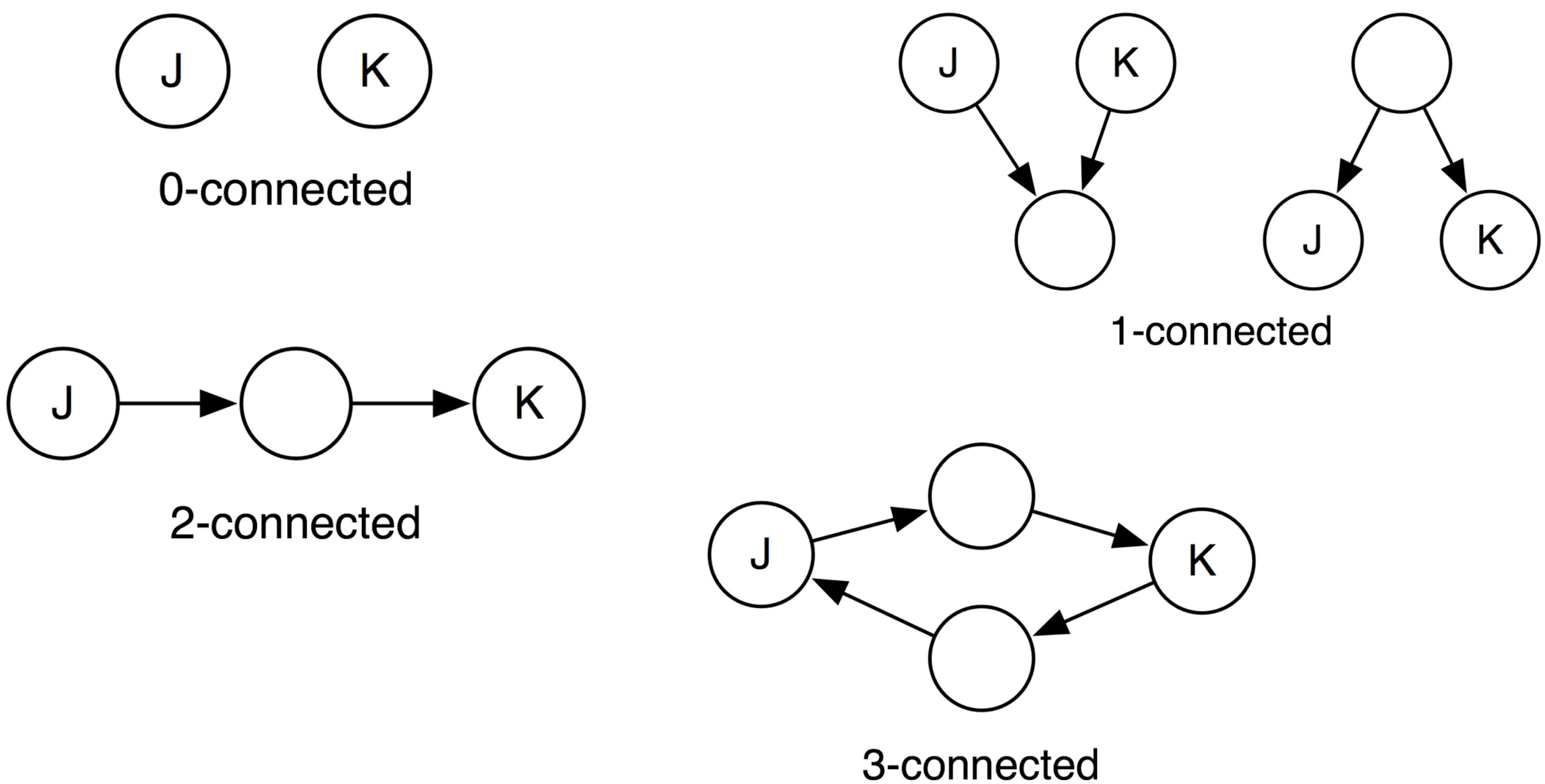
- Telephone example 2 – static distributed interaction
  - **Call forwarding is defined**
    - **Alice has call forwarding to Bob**
    - **Bob has call forwarding to Charlene**
    - **Charlene has call forwarding to Alice**
  - **The call forwarding data is contrary – cannot all be true at the same 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

# Graph connectedness for dynamic interactions

- Make use of n-connectedness in graphs



# Data-data connectedness

- 0-connected – Logically independent
- 2-connected – sub-alternation
- 3-connected – bidirectional – contraries, contradictories and sub-contraries

# Dynamic, single processor interactions

- Six potential pairs interact
  - **Combination pairs of: data, events and threads**
- Each interaction can exhibit 4 different graph connectedness attributes
- Result is 24 sub-categories for these interactions

# Dynamic, single processor interactions – 2

- Examples
  - 1-connected data-data
    - Two or more data items are input the same action
  - 2-connected data-data
    - When a data item is used in a computation
  - 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 – 3

- 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 the thread nets need to be resolved when they interact**
- 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 – 2

- 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**
    - $EI \neq \emptyset \quad \oplus \quad EO \neq \emptyset$
  - **2-connected – only through data places**
    - $EO2 \cap EI1 = \emptyset \quad \oplus \quad EO1 \cap EI2 = \emptyset$
  - **3-connected – only through data places**
    - $EO2 \cap EI1 = \emptyset \quad \wedge \quad EO1 \cap EI2 = \emptyset$

# 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 input & output 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**

## 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**
- 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 finite state machines grow exponentially in size and complexity**
  - **May be difficult to rationalize initial marking**
  - **Have mutual exclusion**
    - **Contraries**
  - **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**

# 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
  - **For a non-deterministic system it may be difficult to demonstrate different output**
    - **Process P non-deterministically chooses at every step whether to output an ‘a’ or a ‘b’**
    - **Process Q non-deterministically chooses once whether to output all ‘a’s or all ‘b’s**

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

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

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

$$Qa = (a \rightarrow Qa)$$

$$Qb = (b \rightarrow Qb)$$

# Formal definition of determinism

- P is deterministic $\leftrightarrow \forall s : \mathit{traces}\,(P) \bullet$
  $$X \in \mathit{refusals}\,(P / s) \leftrightarrow X \cap (P / s)^1 = \{\}$$
  $$P^1 = \{\, e \mid \langle\, x \,\rangle \in \mathit{traces}\,(P) \,\}$$
  - **A system is deterministic if at every step the system never refuses to engage in any external event appropriate at that step**
  - **$P^1$ 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**
    - **An internal event**
      - not stated in the textbook
- 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**

# On non-determinism – 2

- 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**
- 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**

# Client / Server testing

- The 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**
- Lots of possibilities for finger pointing takes place when things go wrong

# Client / Server testing – 2

- Much of the system is stable
  - **Should testing be needed**
    - **Use functional testing – no source text**

# Client / Server testing – 3

- Interesting part is the GUI
  - Consists of multiple windows that need to synchronized
    - Communicating sequential processors (FSMs)
  - All events are port events
  - Have dynamic interactions across multiple processors
  - Use operational profiles
  - Test individual threads
  - Then test thread interaction
    - Big problem if there are multiple clients such as shared bank accounts