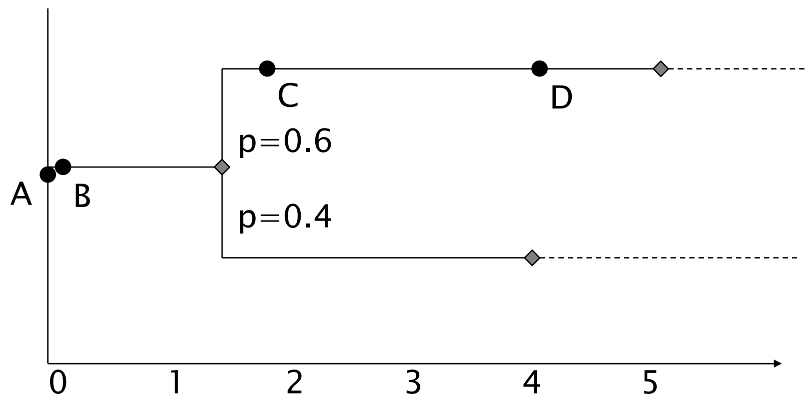
Formal Conformance Testing 2006

Lecture 7
12th Oct 2006

Traces and specifications

- Trace = set of events + end time stamp
  - Event = message + time stamp
  - Prefix, extension, snapshot
- Specification ≅ set of valid traces
  - Prefix-closed
  - Serial

Execution sketch



Test step disjointness

- For any i, s, T, and T_1 and T_2 it must hold that if $T_1 \neq T_2$ and
- $\xi(i, s, T)[T_1] > 0$ and
- $\xi(i, s, T)[T_2] > 0$, then
- $T_1 \not\prec T_2$, and
- $T_2 \not\prec T_1$
- A technical convenience

Progressivity

- There does not exist an infinite sequence T_1, T_2, T_3, … and a constant $K \in \mathbb{R}$ such that
- $\xi(i, s, T_i)[T_{i+1}] > 0$ for all i, but such that for all $T_i = <E_i, t_i>$ it holds that $t_i < K$.

Choice of ξ

- We have defined properties of ξ, not the function itself
- The particular choice for ξ depends on
  - the set of implementations $\mathbb{I}$,
  - the set of testing strategies $\mathbb{T}$, and
  - the desired structure of test steps.

Trace probabilities

- Let P[i, s, T] denote the probability of observing T as a prefix of a long enough trace when strategy s is executed against implementation i
- Idea is to compute the product of the preceding test step probabilities

Trace probabilities

- Random experiment:
- Implementation i and a testing strategy s fixed
- A trace prefix T* has fixed, T*=<E,K>
- s is executed against i many times, yielding traces T_1=<E_1,t_1>, T_2=<E_2,t_2>, …, such that for all n, $t_n > K$
- What is the probability that for a uniformly chosen n, T_n[K] = T*?
  - X[t] is that prefix of X whose end time stamp is t

Solution

- Traces that end at test step boundaries are easy: compute product probability
- Traces that end at non-boundaries require an extra construct

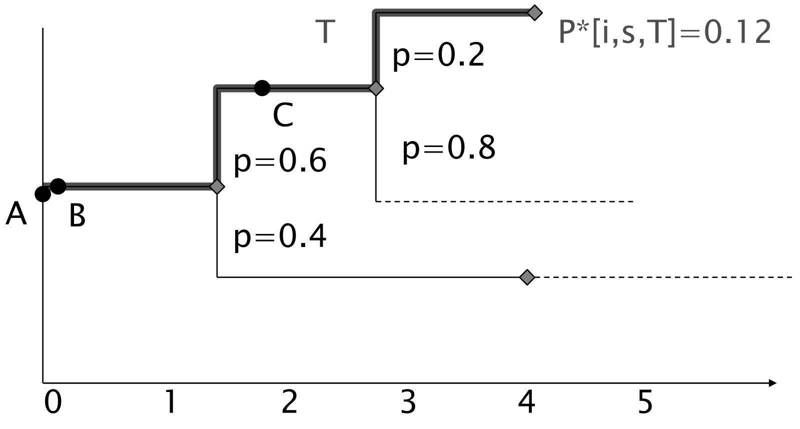
Step 1: Traces at test step boundaries

- Denote by P*[i,s,T]:

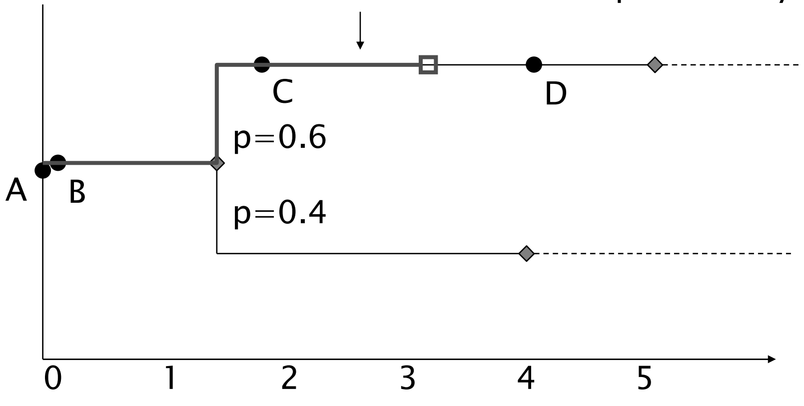
$$\max_{T_1,\ldots,T_n}: \prod_{i\in[1,\, n-1]} \xi(i,s,T_i)[T_{i+1}]$$
$$\text{where } T_1=\epsilon \text{ and } T_n=T$$

- P*[i,s,T] is the compound probability for trace T, if T "happens" at test step boundary

Sketch



Problem



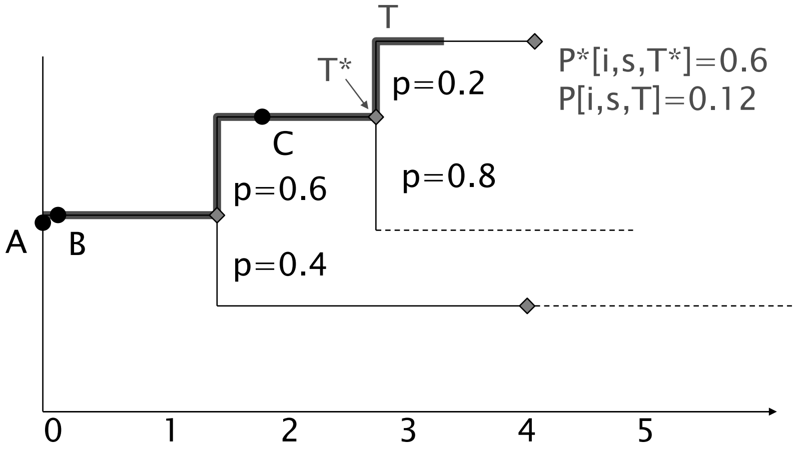
Step 2: traces at non-boundaries

- Denote by P[i,s,T]

$$P^*[i,s,T^*] \times \left(\sum_{T':T \preccurlyeq T'} \xi(i,s,T^*)[T']\right)$$

where T* is the longest prefix of T such that P*[i,s,T*] > 0

Sketch



Sanity checks

- If P*[i,s,T] > 0,
  - then P[i,s,T] = P*[i,s,T].
  - Ok.
- If P[i,s,T] = 0 (trace T cannot be produced),
  - there still exists the greatest prefix T* of T such that P*[i,s,T*] > 0.
  - Every test step succeeding T* must result in a trace differing from T — ok.

Sanity check 1 memo

- Assume P*[i,s,T]>0
- Note T* = T
- P[i,s,T] =

  $$P^*[i,s,T] \times \left(\sum_{T':T \preccurlyeq T'} \xi(i,s,T)[T']\right)$$
- The sum yields one

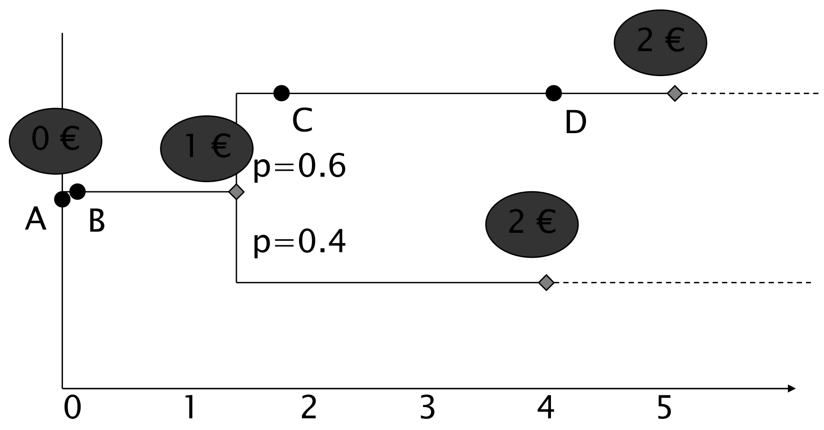
Execution summary

- ξ defines execution semantics
- Properties for ξ
  - Gives probability distribution over traces
  - Test step = trace extension
  - Test step disjointness
  - Progressivity
- However, no concrete structure
- P[i,s,T] is the probability of producing trace T when s is run against i
  - Hides test steps

Why test steps?

- 1 test step =
  - unit of testing <u>cost</u>
  - unit of <u>benefit</u>
- Testing can be stopped between test steps, but not during them
  → stopping criteria
- Technical construct for describing arbitrarily long executions without the concept of “an infinite trace” (there is no such concept here)

Cost or benefit



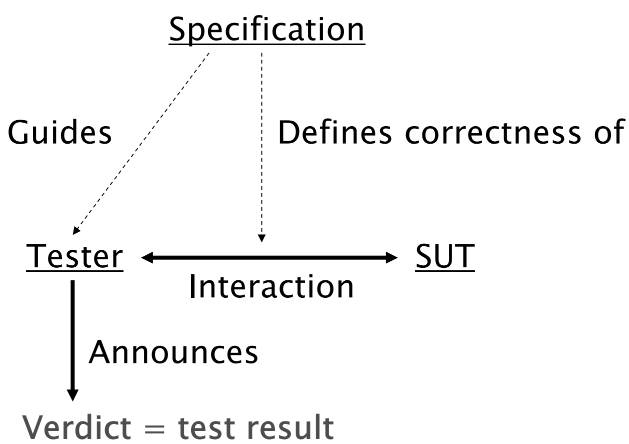
Measuring the "size" of a trace

- Temporal length = end time stamp
- Size of event set
- Number of test steps used to produce

Testing of Concurrent Systems 2004

Lecture 8
28th Sep 2004

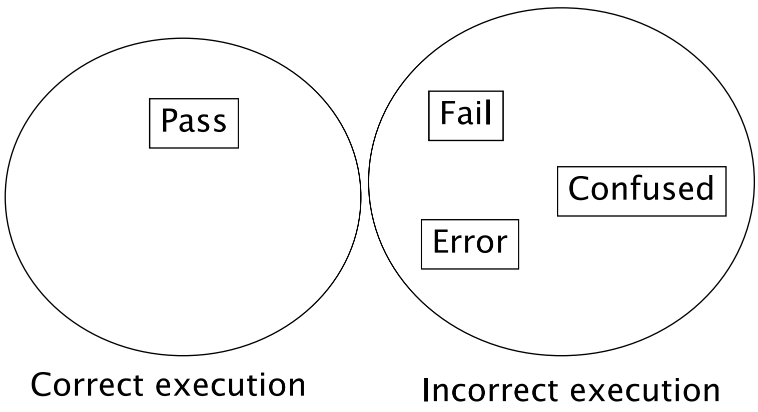
Verdicts



Verdicts

- Pass
- Fail
- Error
- Confused

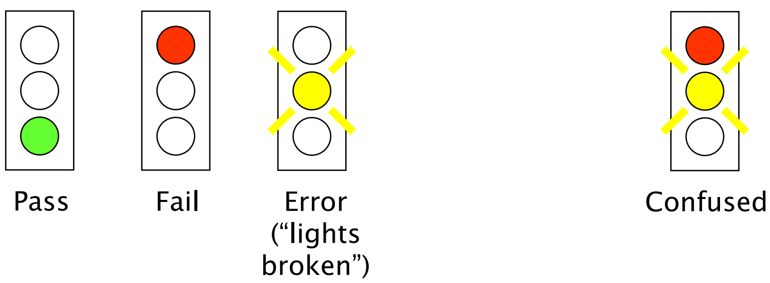
Verdicts



Verdicts explained

Verdict	*Explanation*
Pass	System under test has behaved correctly
Fail	System under test has behaved incorrectly
Error	Tester has behaved incorrectly
Confused	Fail-and-Error, result produces by an ambiguous specification (a special corner case)

Verdicts as traffic lights



Calculating verdict

- Verdict is calculated from a trace T and a specification S

verdict(T,S) $\in$ { pass, fail, error, conf }

Pass verdict

verdict(T,S) = pass

if and only if

T $\in$ Tr(S)

Other verdicts

- Hence, $T \notin Tr(S)$ implies

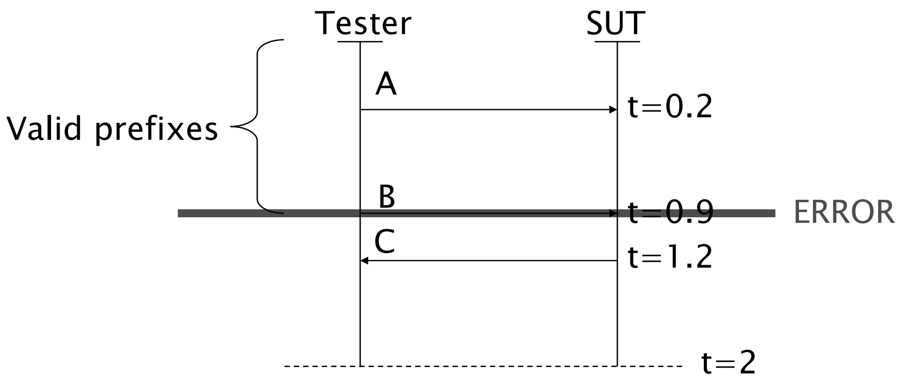
  $verdict(T,S) \in \{ fail, error, conf \}$

- There is one verdict for $T \in Tr(S)$, and three for the other case

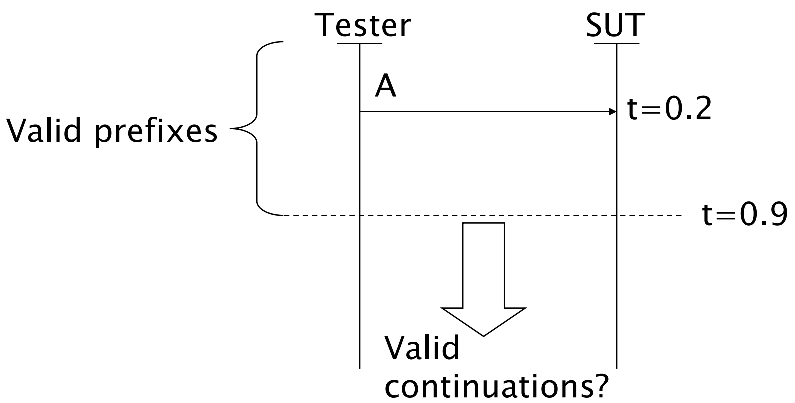
Other verdicts

- The problem: how to classify the cases $T \notin Tr(S)$ into
  - errors of the SUT (→ fail),
  - errors of the tester (→ error),
  - and those cases where the erring party cannot be defined (→ confused)?

Solution sketch (1)



Solution sketch (2)



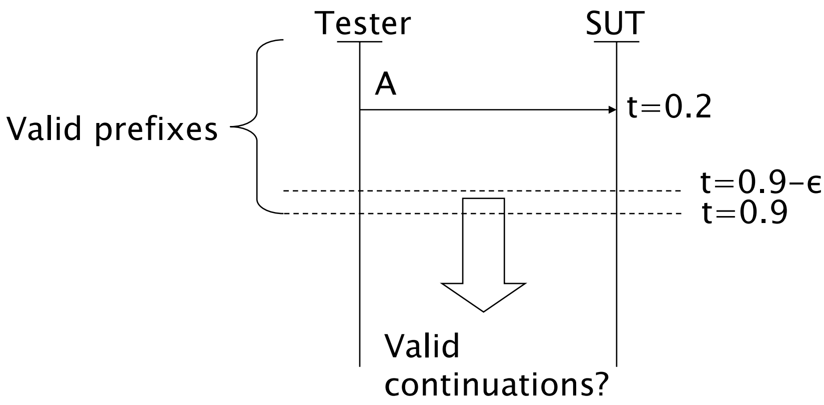
Solution sketch (3)

- All valid continuations differ from T first at input events?
  → error [tester error]
- All valid continuations differ from T first at output events?
  → fail [SUT failure]
- Otherwise
  → confused [unclear]

Technicality

- The set of end time stamps for the valid prefixes of T can be either open or closed at the upper boundary
- Open set requires basically a limit construct (as usual)

Solution sketch (4)



Details

- Assume $T \notin Tr(S)$
- Let $V = Tr(S) \cap Pfx(T)$
  - Note: $\epsilon \in V$
  - This is the set of valid proper prefixes of T
- Let $K = \{ t \mid \exists E: \langle E, t \rangle \in V \}$
- K is either
  - closed: $[0, t]$, or
  - open: $[0, t)$.
  - It is the set of end time stamps in V.

Example (closed set)

- Tr(S) =

$\cup$ { Pfx(<{<A,t'>},t>) | t$\in$[2,∞), t'$\leq$ 1 }
- T = <$\emptyset$, 10>
- Note that T $\notin$ Tr(S)
- V = { <$\emptyset$, t> | t $\leq$ 1 }
- K = [0, 1]
- Especially <$\emptyset$, 1> is in V, because <{<A,1>, 1.1> is valid

Example (open set)

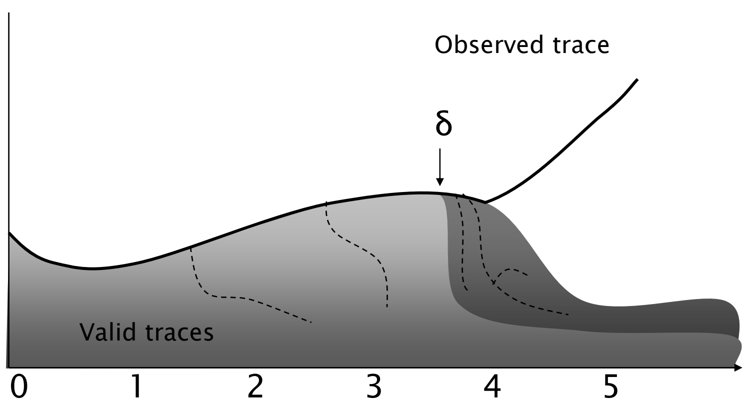
- Tr(S) =

$\cup$ { Pfx(<{<A,t'>},t>) | t$\in$[2,∞), t'<1 }
- T = <$\emptyset$, 10>
- Note that T $\notin$ Tr(S)
- V = { <$\emptyset$, t> | t < 1 }
- K = [0, 1)
- Especially <$\emptyset$, 1> is not in V, because for any t' < 1, event <A,t'> should belong to the event set at time 1

Details continued

- Choose $\delta \in K$ (note: $0 \in K$ always, so K is not empty)
- Let X_δ denote the set of all valid extensions of $T[\delta]$ beyond the end time stamp of T
- $T[\delta]$ is the prefix of T with end time stamp δ

Sketch



Copyright © Antti Huima 2004-06. All Rights Reserved.

Details continued

- For every T' in X_δ, T' differs from T and $\Delta(T,T')$ is defined
- For every T', denote by α $T|_{\Delta(T,T')}$ if not τ
  - Otherwise denote by α $T'|_{\Delta(T,T')}$
  - Note: $\underline{\alpha \text{ can not be } \tau}$
- Let D_δ be the union of all α
- D_δ lists those events on which valid extensions of T[δ] differ from T first

Details continued

- Assume there exists $\delta \in K$ such that $D_\delta \subseteq \Sigma_{in}$
  - Tester failure → error
- Assume there exists $\delta \in K$ such that $D_\delta \subseteq \Sigma_{out}$
  - SUT failure → fail
- Otherwise
  - undefined → confused

Details continued

- If K is closed, we can always choose $\delta = \max K$
- If K is open, we must choose a δ "close enough" the upper bound of K
  - $(\sup K) - \epsilon$ for $\epsilon > 0$

Disjointness

- $D_\delta \subseteq \Sigma_{in}$ and $D_\delta \subseteq \Sigma_{out}$ are disjoint conditions, because
  - $\delta \leq \epsilon$ implies $D_\epsilon \subseteq D_\delta$
  - D_δ is always non-empty
  - Σ_{in} and Σ_{out} are disjoint

Summary

- Is $T \in Tr(S)$?
  - Verdict is ”pass”
- Else
  - Does there exists $\delta \in K$ such that $D_\delta \subseteq \Sigma_{out}$?
    - Verdict is ”fail”
  - Otherwise, does there exists $\delta \in K$ such that $D_\delta \subseteq \Sigma_{in}$?
    - Verdict is ”error”
  - Otherwise verdict is ”confused”

Examples

- Spec: “System must send out X before time 2”
- Observed trace $<\emptyset, 2>$
- Verdict?

Examples

- Spec: “System must send out X at latest at time 2”
- Observed trace $<\emptyset, 2>$
- Verdict?

Examples

- Spec: “System must send out X before time 2 if it receives Y before 1”
- Observed trace $<\emptyset, 2>$
- Verdict?

Examples

- Spec: “System must send out X before time 2 if it receives Y before 1”
- Observed trace <{<Y,0.5>}, 2>
- Verdict?

Examples

- Spec: “System must send out X before time 2 if it receives Y before 1”
- Observed trace <{<Y,1.5>}, 2>
- Verdict?

Examples

- Spec: “System must receive clock signal every 1 second starting from t=1”
- Observed trace <{<clock,1>}, 1.5>
- Verdict?

Examples

- Spec: “System must receive clock signal every 1 second starting from t=1”
- Observed trace <{<clock,1>}, 2>
- Verdict?

Copyright © Antti Huima 2004-06. All Rights Reserved.

Examples

- Spec: “System must receive clock signal every 1 second starting from t=1”
- Observed trace $<\{<clock,1>\}, 2.5>$
- Verdict?

Examples

- Spec: “Starting from t=1, on every second system must either send or receive X”
- Observed trace $<\emptyset, 10>$
- Verdict?

Discussion

- These definitions are abstract and intensional
- They do not involve and <u>algorithm</u>
- They are given directly for trace sets making them <u>universal</u>

Formal Conformance Testing 2006

Lecture 9

2nd November 2006

Conformance?

- What does it mean that a system conforms to a specification?
  - System functions as specified
  - System passes all tests
  - Which "all" tests?
  - System passes every "test" that is "correct"
  - What is "a test"? What is "a correct test?"

What is "a test"?

- A test = ?
  - a specific testing strategy
  - a specific test execution trace
  - a specific tester
  - a specific tester execution
- A correct test = ?
  - A test execution trace with verdict ≠ ERROR
  - A testing strategy or tester that "never works illegally"
    - What does this mean?

Correct testing strategies

- A testing strategy s is correct with respect to a specification S if for any implementation i:

  $P[i,s,T] > 0 \Rightarrow \text{verdict}(T, S) \neq \text{ERROR}$
- Denote by CT(S) the set of all correct testing strategies with respect to S

Synthetic correct strategies

- A correct testing strategy can be (informally) constructed by the following loop:
  - Guess the next action (send/wait) so that a valid trace extension will result
  - Execute the chosen action, observing the actions of the SUT
  - Restart loop
- More on this later!
- Shows that correct testing strategies exist
  - Possible because of the seriality of valid set of traces
- In real life computationally intensive

Correct strategies ctd

- If we assume these synthetic strategies belong to the set of available testing strategies…
- … then all correct and failing behaviours (at least prefixes of) of a system can be observed.

Classifying traces

- Let i be an implementation and S a specification
- For every trace T, one of the following is true:
  - There exists $s \in CT(S)$ such that
    $P[i, s, T] > 0$
  - There exists s, but none in CT(S), such that
    $P[i, s, T] > 0$
  - There does not exist any s such that
    $P[i, s, T] > 0$

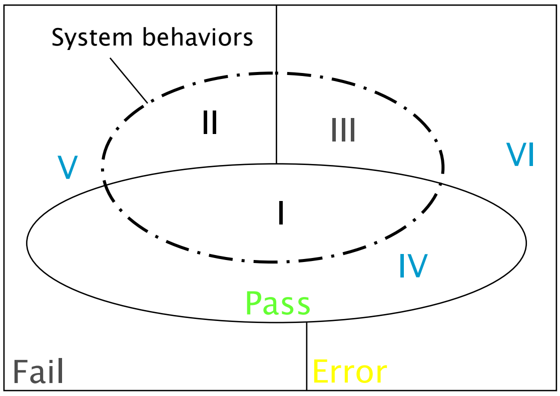
Trace taxonomy ctd

- Furthermore, for every trace T it holds that verdict(T, S) is one of PASS, FAIL, ERROR
- We ignore ambiguous specifications (→ verdict CONFUSED) for now

Matrix for a trace T

Verdict → Condition	PASS	FAIL	ERROR
$\exists s \in CT(S)$: $P[i, s, T] > 0$	Possible		Not possible (def. correct strategy)
$\exists s$: $P[i, s, T] > 0$ $\forall s'$: $P[i, s', T] > 0$ $\Rightarrow s' \notin CT(S)$	Not possible (synthetic testers)		Possible
$\nexists s$: $P[i, s, T] > 0$	Possible (impl. restriction)		

Trace map of a general system



CORRECT AND INCORRECT TESTERS

I: correct, producible behaviour

II: incorrect, producible

INCORRECT TESTERS ONLY

III: producible behaviour against malfunctioning environment only

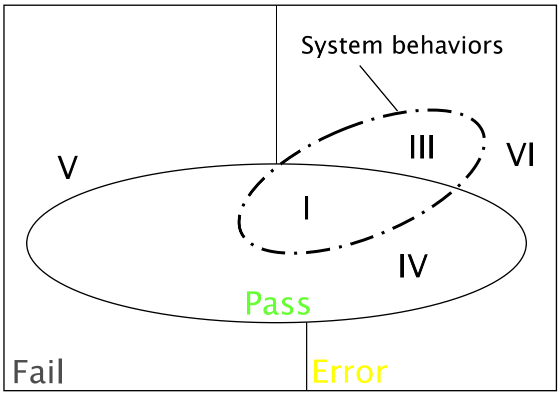
NO TESTERS AT ALL

IV: correct behaviour not implemented

V: incorrect behaviour not implemented

VI: behaviour involving malfunctioning environment that has not been implemented

Trace map of a correct system



CORRECT AND INCORRECT TESTERS

I: correct, producible behaviour

INCORRECT TESTERS ONLY

III: producible behaviour against malfunctioning environment

NO TESTERS AT ALL

IV: correct behaviour not implemented

V: incorrect behaviour not implemented

VI: behaviour involving malfunctioning environment that has not been implemented

Correct system ctd.

- If we restrict ourselves to correct testers, then
- all behaviour that can be generated is included within the set of valid traces Tr(S). [Note: assumed that system is "correct".]

Execution against correct strategies

- Define set of execution traces

  $$\mathrm{ETr}(i) = \{ T \mid \exists s : P[i,s,T] > 0 \}$$

- Define now

  $$\mathrm{ETr}(i, S) = \{ T \mid \exists s \in \mathrm{CT}(S) : P[i,s,T] > 0 \}$$

- Here CT(S) is the set of testing strategies correct with respect to S
- Note $\mathrm{ETr}(i, S) \subseteq \mathrm{ETr}(i)$ for all S

Continued…

- We have now eliminated the ERROR verdict
- Suppose for all $s \in CT(S)$,

  $P[i, S, T] > 0$ implies $verdict(T, S) \neq$ FAIL
- Then (assuming unambiguous specifications),

  $P[i, S, T] > 0$ implies $verdict(T, S) =$ PASS
- Hence, <u>$ETr(i, S) \subseteq Tr(S)$</u>

Conclusion

- We have thus reduced the conformance of an implementation i to a specification S to the equation

  <u>$ETr(i, S) \subseteq Tr(S)$</u>
- This is <u>the</u> underlying notion of conformance in the known theory of formal conformance testing

Conclusion ctd.

- Conformance = trace inclusion
  - Traces generated by implementation are included in those generated by specification
  - Incorrectly generated/out-of-specification traces must be excluded
  - Note: no explicit mention of single testing strategies above!

Implications

- Quantifying over all testers leads to simple trace set inclusion
- This trace set inclusion can be also checked for <u>directly</u> under suitable conditions → model checking
- Thus formal conformance testing = "partial model checking"