Tester versus Bug: A Generic Framework for Model-Based Testing via Games

Petra van den Bos & Marielle Stoelinga

Radboud University & University of Twente, the Netherlands

GandALF 2018

STW
Enabling new technology

iCIS | Software Science
Radboud University

Testing is playing a game

- In each system state:
  - Tester: chooses input
  - System Under Test (SUT): chooses output
  - Next state based on chosen input and output
- Resembles two-player game!

Why testing?

- Testing is a widely practiced method
- Testing is expensive: 30% of development cost
- Establish link between model-based testing and game theory

Linking games and model-based testing

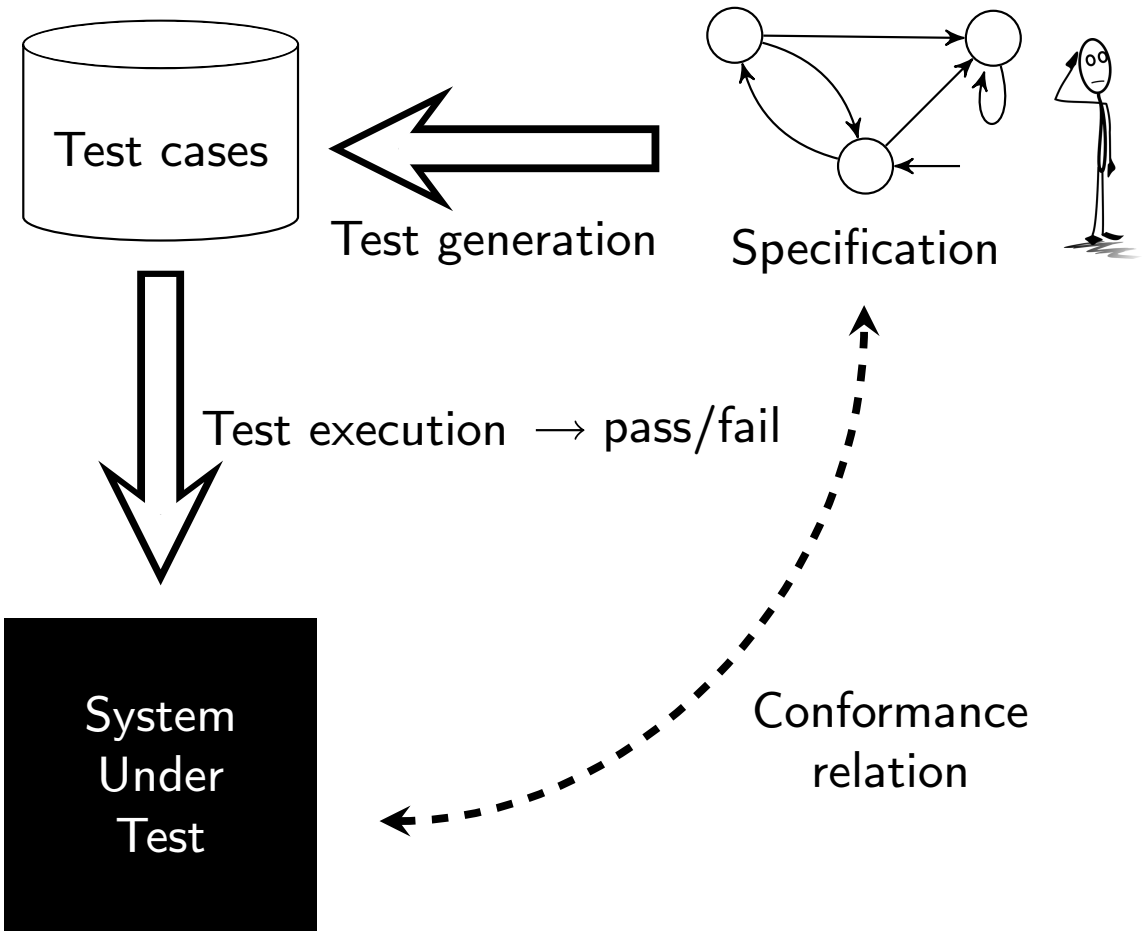
- Reuse techniques from game theory in model-based testing!
- Example use case: extract test case from winning strategy

Linking games and model-based testing

- Reuse techniques from game theory in model-based testing!
- Example use case: extract test case from winning strategy
- Existing approaches start from game arenas
- But: testers think in terms of system specifications

iCIS | Software Science
Radboud University

Model-Based Testing



Test cases and conformance relation in games

- Contributions:
  1. Explicit encoding of specifications as game arenas

Test cases and conformance relation in games

- Contributions:
  1. Explicit encoding of specifications as game arenas
  2. Encoding includes assumptions on interaction of tester and SUT

Test cases and conformance relation in games

- Contributions:
  1. Explicit encoding of specifications as game arenas
  2. Encoding includes assumptions on interaction of tester and SUT
  3. How test cases relate to strategies

Test cases and conformance relation in games

- Contributions:
  1. Explicit encoding of specifications as game arenas
  2. Encoding includes assumptions on interaction of tester and SUT
  3. How test cases relate to strategies
  4. How conformance relates to alternating trace inclusion

Game arenas

A *game arena* is a tuple $G = (Q, q_0, \mathrm{Act}_1, \mathrm{Act}_2, \Gamma_1, \Gamma_2, \mathit{Moves})$ where, for $i = 1, 2$:

- Q is a finite set of states,
- $q_0 \in Q$ is the initial state,
- Act_i is a finite and non-empty set of Player i actions,
- $\Gamma_i : Q \to 2^{\mathrm{Act}_i} \setminus \emptyset$ is an enabling condition, and
- $\mathit{Moves} : Q \times \mathrm{Act}_1 \times \mathrm{Act}_2 \to 2^Q$ is a function determining the moves

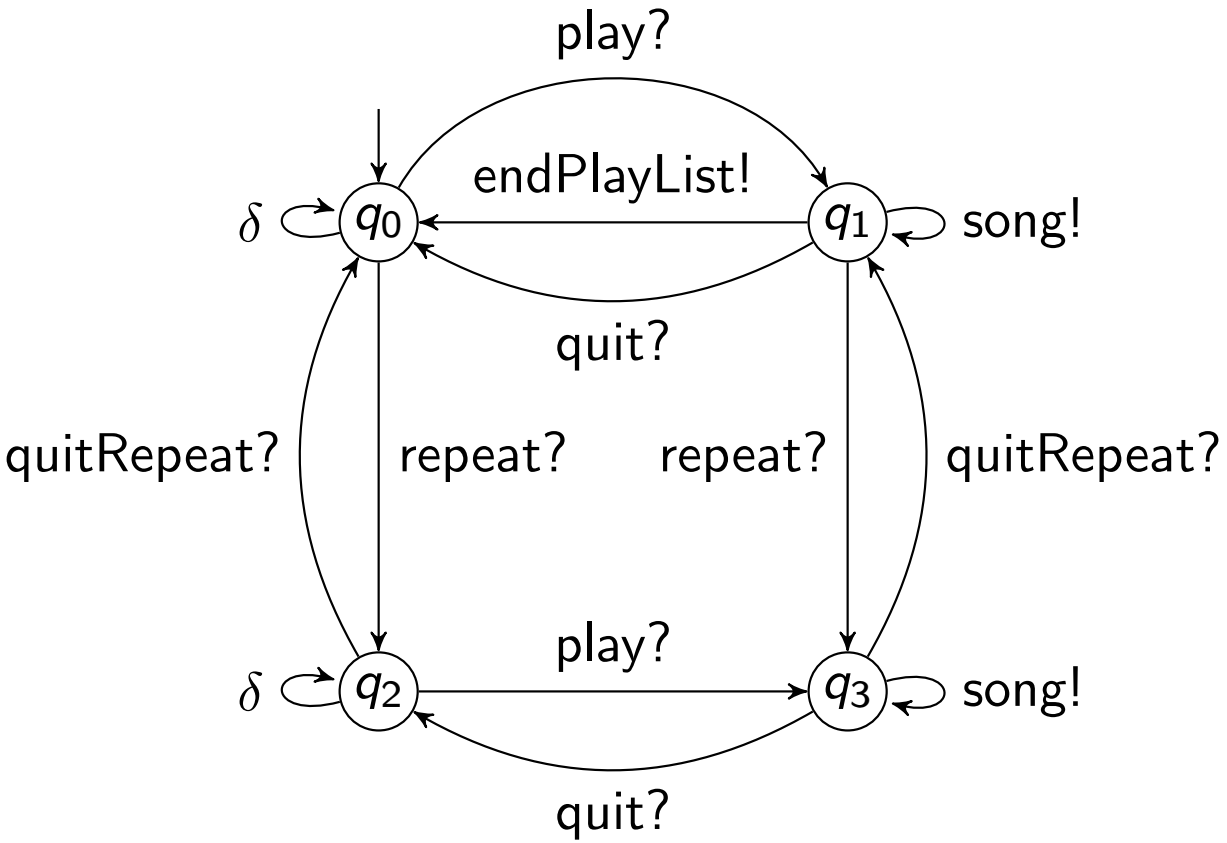
iCIS | Software Science
Radboud University

Specifications

A *suspension automaton* (SA) is a tuple $\mathcal{A} = (Q, L_I, L_O^\delta, T, q_0)$ where

- Q is a non-empty finite set of states,
- L_I is a finite set of input labels,
- $L_O^\delta = L_O \cup \{\delta\}$ with L_O a finite set of output labels, $\delta \notin L_O$, and $L_I \cap L_O^\delta = \emptyset$,
- $T : Q \times (L_I \cup L_O^\delta) \rightharpoonup Q$ is a partial transition function , and
- $q_0 \in Q$ is an initial state.

Specification of an MP3 player



1. Encoding a specification as a game

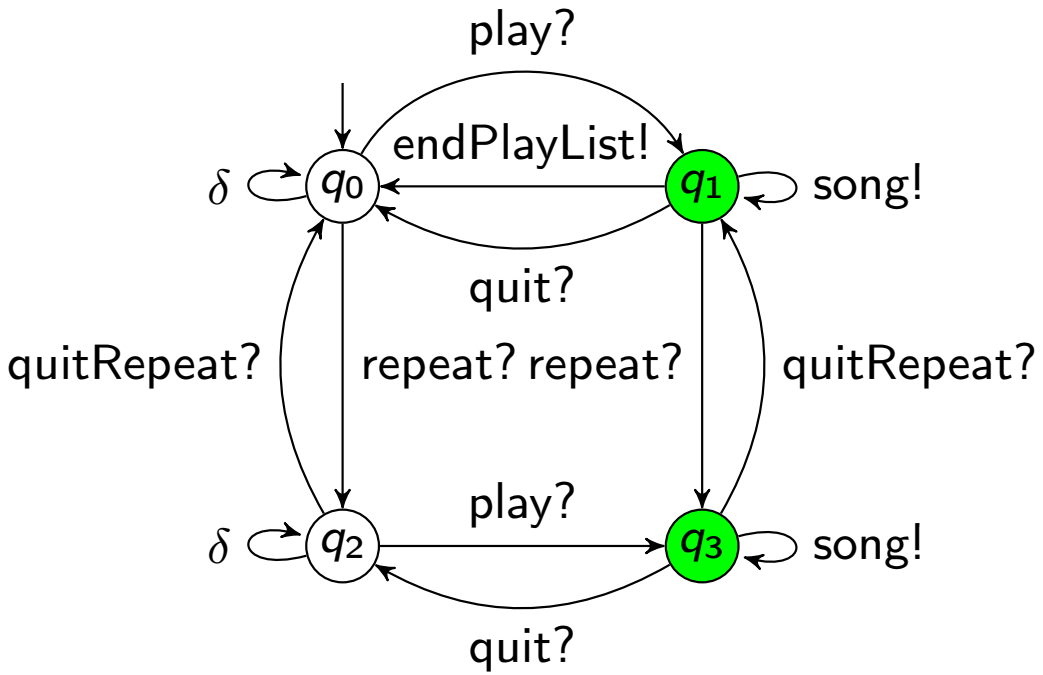
Let $\mathcal{A} = (Q, L_I, L_O^\delta, T, q_0)$ be an SA. The *game arena underlying* $\mathcal{A}$ is defined by $G_\mathcal{A} = (Q_\perp, (q_0, 1), \mathrm{Act}_1, \mathrm{Act}_2, \Gamma_1, \Gamma_2, \mathit{Moves})$ where:

- $Q_\perp = Q \times \{1, 2\}$
- $\mathrm{Act}_1 = L_I \cup \{\theta\}$ and $\mathrm{Act}_2 = L_O^\delta$,
- $\Gamma_1((q, i)) = \mathit{in}(q) \cup \{\theta\}$ and $\Gamma_2((q, i)) = \mathit{out}(q)$,
- the function $\mathit{Moves} : Q_\perp \times \mathit{Act}_1 \times \mathit{Act}_2 \to 2^{Q_\perp}$ encodes one of the different test assumptions.

Auxilary definitions for SAs:

- $\mathit{in}(q) = \{a \in L_I \mid T(q, a)\!\downarrow\}$,
- $\mathit{out}(q) = \{x \in L_O^\delta \mid T(q, x)\!\downarrow\}$, and
- An SA is non-blocking: $\forall q \in Q : \mathit{out}(q) \neq \emptyset$.

Test assumptions



- Input-eager: the tester gets priority in providing inputs
- Output-eager: the SUT gets priority in providing outputs
- Nondeterministic: both can happen, no guarantees

2. Moves for test assumptions

- $Moves : Q_\perp \times Act_1 \times Act_2 \to 2^{Q_\perp}$
- *Moves* for the Input-Eager test assumption:

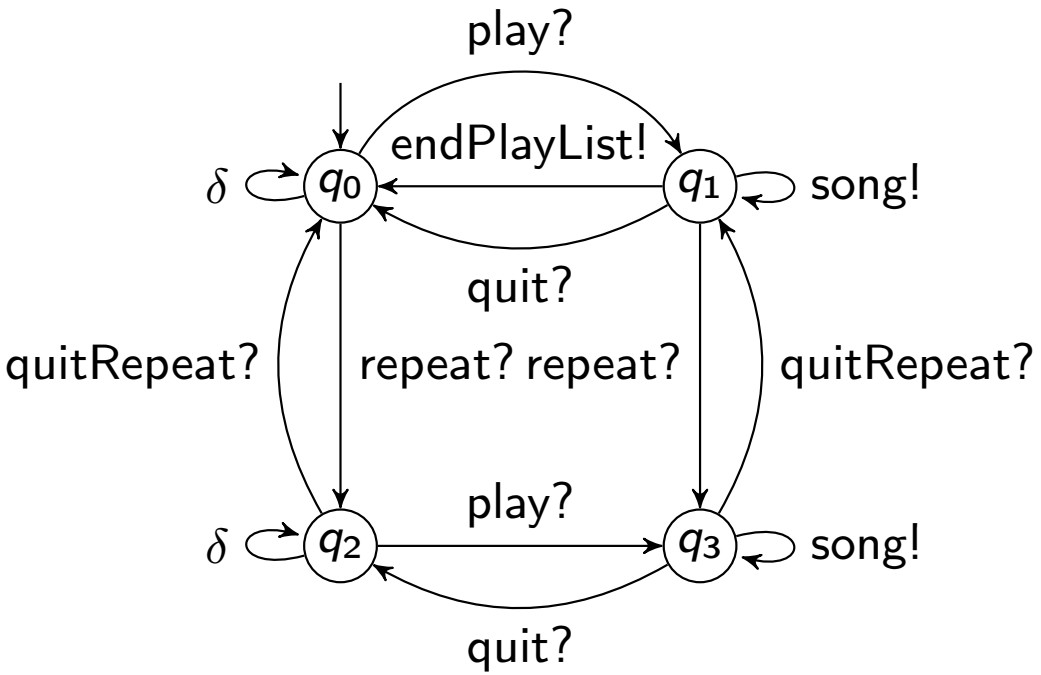
$$Moves_{IE}((q,i),a,x) = \begin{cases} \{(T(q,a),1)\} & \text{if } a \neq \theta \\ \{(T(q,x),2)\} & \text{otherwise} \end{cases}$$

- *Moves* for the NonDeterministic test assumption:

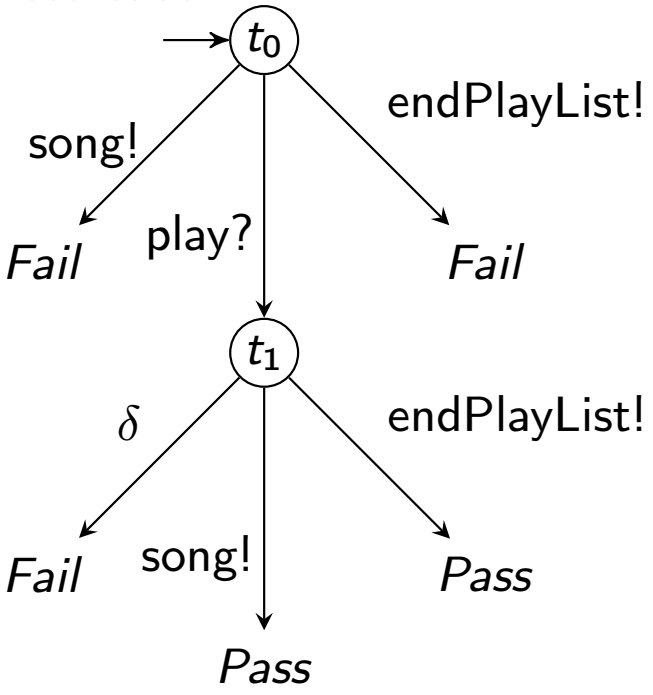
$$Moves_{ND}((q,i),a,x) = \begin{cases} \{(T(q,x),2)\} & \text{if } a = \theta \\ \{(T(q,a),1)\} & \text{if } x = \delta \\ \{(T(q,a),1),(T(q,x),2)\} & \text{otherwise} \end{cases}$$

Test cases

Specification:



Test case:



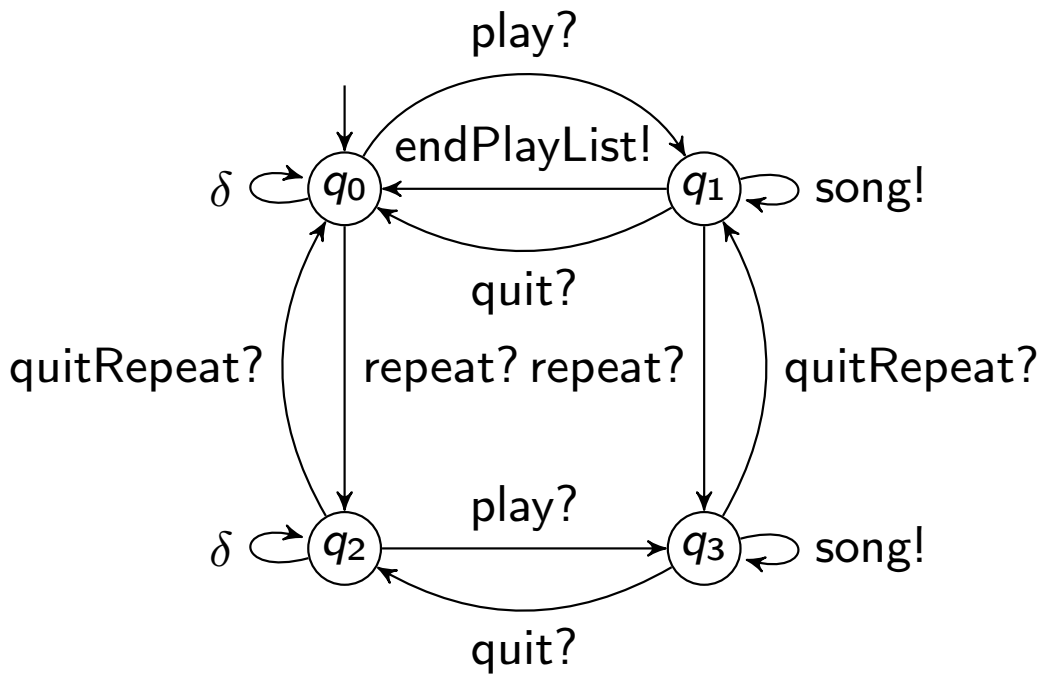
Plays and strategies

- A *play* of a game is an infinite sequence:
  $\pi = p_0\langle a_0, x_0\rangle p_1\langle a_1, x_1\rangle p_2 \ldots$
- A player 1 strategy in G is a function: $\sigma_1 : \mathit{Pref}(\Pi(G)) \to \mathrm{Act}_1$

iCIS | Software Science
Radboud University

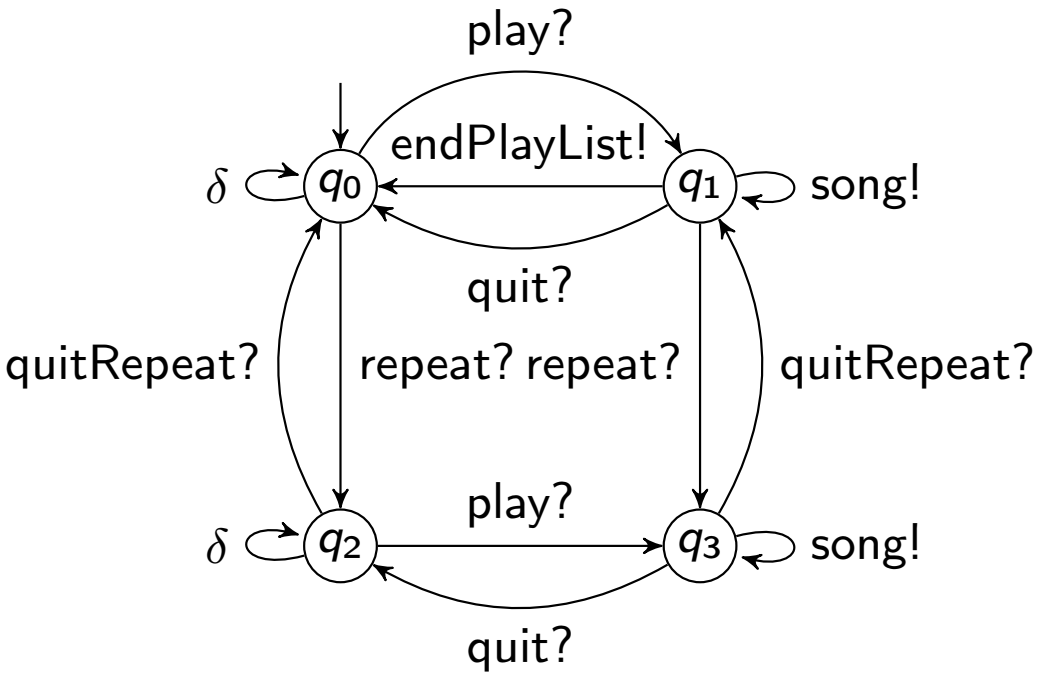
Plays and strategies

- A *play* of a game is an infinite sequence:
  $\pi = p_0\langle a_0, x_0\rangle p_1\langle a_1, x_1\rangle p_2 \ldots$
- A player 1 strategy in G is a function: $\sigma_1 : \mathit{Pref}(\Pi(G)) \to \mathrm{Act}_1$
- Example play prefix: $\pi = (q_0, 1)\langle \mathsf{play?}, \delta\rangle(q_1, 1)\langle \theta, \mathsf{song!}\rangle(q_1, 2)$
- $\mathit{trace}(\pi) = \mathsf{play?song!}$

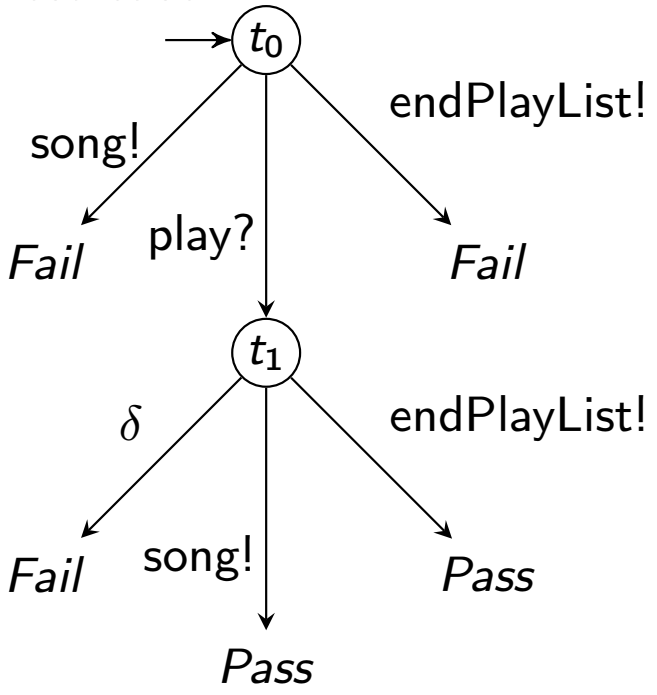


From test case to strategy

Specification:



Test case:



$$\sigma_1(\pi) = \begin{cases} \text{play?} & \text{if } \mathit{trace}(\pi) = \epsilon \\ \theta & \text{if } \mathit{trace}(\pi) = \text{play?} \\ \text{stop?} & \text{otherwise} \end{cases}$$

iCIS | Software Science
Radboud University

4. Alternating trace inclusion and ioco

Theorem: $\mathcal{I}$ ioco $\mathcal{S} \iff G_{\mathcal{I}} \sqsubseteq_2 G_{\mathcal{S}}$

Conclusions

Contributions of the paper:

1. Explicit encoding of specifications as game arenas
2. Encoding includes assumptions on interaction of tester and SUT
3. How test cases relate to strategies
4. How conformance relates to alternating trace inclusion