Incremental Integration Testing of Concurrent Programs

P.V. Koppol, R.H. Carver and K.C. Tai

Presented by

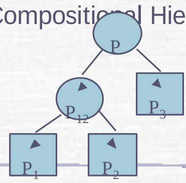
Chuk Yang Seng

Roadmap

- Select test sequences for concurrent programs
  - Labeled transition systems (LTS)
  - Problems
- Incremental Approach
  - Annotated labeled transition system (ALTS)
  - Reduction Algorithms
- Coverage Criteria

Introduction

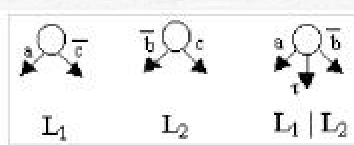
- Test case
  - A test sequence – a sequence of actions performed by the concurrent processes.
- Model: Compositional Hierarchy



Introduction

- Labeled Transition System
  - Node – state of a process
  - Edge
    - **Actions performed during state transition**
    - **Interactions between processes**

Introduction - LTS



- Composition of LTS -> L_g (reachability graph)

Introduction

- To select test sequences
  - Select a set of paths from reachability graph.
  - For each selected paths, derive one or more inputs and force deterministic executions according to the path – deterministic testing.

Introduction

- Problem
  - State explosion: number of states in the reachability graph is exponential to number of processes.

Incremental Reachability Analysis

- Building a reduced LTS L_g^r.
- Reduced LTS must be semantically equivalent to L_g.
- Strong equivalence:
  - 2 LTSs whose behaviors are indistinguishable to an observer, including ??events.

Incremental Reachability Analysis

- Observational equivalence:
  - 2 LTSs whose behavior are indistinguishable when ??events are invisible.
- To build reduced LTS
  - Subsystems are successively composed and simplify.
  - Simplify be removing some ??events.
  - Simpler but observationally equivalent LTS

Incremental Reachability Analysis

- However, paths from the reduced graph cannot be used for deterministic testing.

Annotated LTS

- e-transition (non ?)
  - $(\bar{e}, i, ?)$ process *i* performs this send event and the identifier of the receiver will be determined during synchronization.
  - $(e, ?, j)$ process *j* performs receive event with the identifier of the sender to be determined during synchronization.
- ?-transition
  - Synchronize 2 matching events: (e, i, j)

Annotated TLS

- ALTS reduction:
  - Suppose we have a sequence of ?? transitions:

$$?_1(e_1,i_1,j_1)?\ ?_2(e_2,i_2,j_2)?\ \square\ ?\ ?_m(e_m,i_m,j_m)$$

  - We can collapse them into a single ?? transition: $((e_1,i_1,j_1)?(e_2,i_2,j_2)\square\ (e_m,i_m,j_m))$

Annotated TLS

- Suppose we have an a-transition (a is not equals to ???such that a is preceded and followed by a ?? transition
  - The result of collapsing into a single a-transition is:

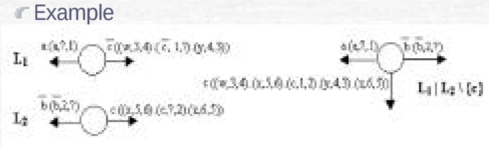
$$((e_1^p,i_1^p,j_1^p)?(e_2^p,i_2^p,j_2^p)\ldots(e_m^p,i_m^p,j_m^p)?(a,?,1)?$$
$$(e_1^s,i_1^s,j_1^s)?(e_2^s,i_2^s,j_2^s)\ldots(e_r^s,i_r^s,j_r^s))$$

Annotated TLS

- Synchronizing:
  - Process 1: $P_1 ?(e,?,1) ?S_1$
  - Process 2: $P_2 ?(\bar{e},2,?) ?S_2$
  - Composite: $P_1 ?P_2 ?(e,2,1) ?S_1 ?S_2$
    - Where P and S are ?-transitions

Annotated TLS

- Example



- ALTS is deterministic if
  - No state that has 2 or more outgoing transitions with same event name/annotation

ALTS Reduction Algorithm

- ALTS A is reduced into a smaller ALTS A'
- A' must satisfied 2 properties:
  - A' must be observationally equivalent to A.
  - Each path of A' must be a path of A.
- 3 procedures:
  - Collapse
  - ??–eliminate
  - Prune

ALTS Reduction Algorithm - Collapse

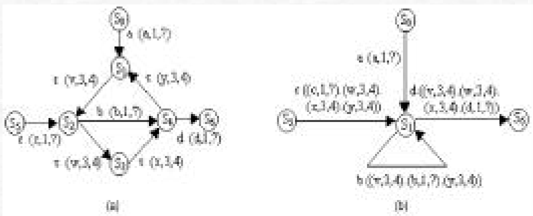
- $?^k$ be a sequence of ?-transitions (length k).
- 2 states are, s_1, s_2 in the same ?-component if:
  - $s_1 ? ?^{k_1} ? s_2$ and
  - $s_2 ? ?^{k_2} ? s_1$

ALTS Reduction Algorithm - Collapse

- Pick 1 state from ?-component.
- Call this the survivor state. The survivor state will remain while we remove the rest.
- Observable transitions are retained.

ALTS Reduction Algorithm - Collapse

- Example:

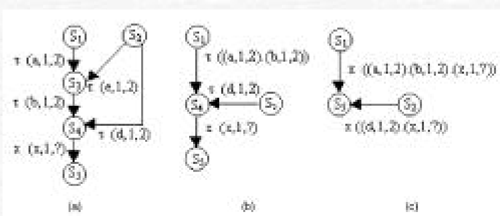


ALTS Reduction Algorithm – ?-Eliminate

- Candidate states (?-states) satisfies
  - All incoming transitions are ?-transitions.
  - One or more outgoing transitions.
  - One or both of:
    - All outgoing transitions are ?-transitions.
    - Source state for each incoming ?-transitions is observationally equivalent to the state.

ALTS Reduction Algorithm – ?-Eliminate

- Example:



ALTS Reduction Algorithm – Prune

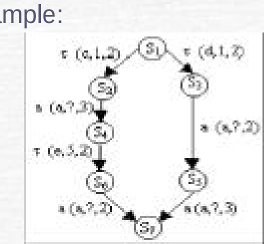
- 2 paths of an ALTS are said to be **externally equivalent** if :
  - They start from the same state.
  - Have the same external behavior (ignoring annotations)
  - Lead to either the same state or to 2 different termination states

ALTS Reduction Algorithm – Prune

- Suppose *s* has e-transitions to *s'* and *s"* and *s'* and *s"* are observational equivalent.
- For every path that starts at *s* and has an e-transition to *s'*, there is at least one externally equivalent path that also starts at *s* and has an e-transition into *s"*.
- Delete one of the transitions.
- After deleting, some other states may become unreachable. These states are also removed.

ALTS Reduction Algorithm – Prune

- Example:



ALTS Reduction Algorithm

- Eliminate all self looping ?? transitions.
- Partition the states with respect to observational equivalence.
  - Compute the transitive closure of ?-transitions.
  - Identify ?-components.
- **Collapse** the ?-components.
- ?**-eliminate**

ALTS Reduction Algorithm

- While(reduced)
  - Prune
  - If (reduced)
    - ?-eliminate
- End while

Bottom Up Incremental Testing

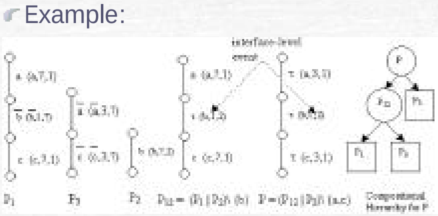
- For intermediate node, N:
  - Synchronizations are at interface level if they occur among immediate children of N.
  - Lower-level synchronizations occur within each immediate child of N.

Bottom Up Incremental Testing

- Bottom up traversal
- At nonleaf node, N:
  - Generate ALTS A_N
  - Select a set T of test paths from A_N
  - Convert T into set T' of test paths for P_N where P_N is the set of processes in P corresponding to ALTSs in node N
  - Use T' to perform deterministic testing of P_N
  - If N is root, terminate. Else reduce A_N to A_N' (such that these 2 ALTS are observationally equivalent)

Bottom Up Incremental Testing

- Example:

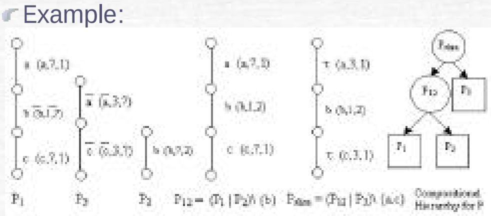


Incremental Testing Using Program Slice

- After constructing A_N we replace each interface-level transition label ?? of A_N with a non-?? label.
- Bottom up traversal and reduction of intermediate nodes until root node is reached.
- The interface level synchronizations of A_N remain in the ALTS.
- The resulting root represents a slice of program P.
- The paths selected focus on the coverage of interface-level transitions of node N.

Incremental Testing Using Program Slice

- Example:



Comparison

- Bottom up
  - Paths generated from an immediate node may not correspond to any paths of global ALTS.
  - Bottom up can be used to test parts of the program.
- Program slice
  - The test paths are generated from a global ALTS.
  - All or nothing: Root node may be too large to be generated and reduced.

Comparison

- Bottom up
  - Test paths generated do not specify a path through the processes in the environment of Ps. The paths include interactions with some of the processes in the environment and they must be simulated by drivers.
- Program slice
  - Test paths include all of the processes in the program, including environment.

Coverage Criteria

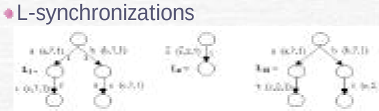
- Property:
  - C – incremental coverage criterion.
  - T – a set of test paths.
  - If C is applied to the reduced ALTSs and T satisfies C, then T would also satisfy C if C were applied to the unreduced ALTS.

Some Coverage Criteria

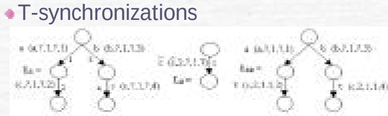
- All paths:
  - Cover all paths of an ALTS at least once.
- All-proper-paths:
  - Proper-path is a path that does not contain any duplicate states, except the first and last may be duplicated once.
  - Cover all proper-paths of an ALTS at least once.
- All transitions:
  - Cover all transitions of an ALTS at least once.
- All states:
  - Cover all states of an ALTS at least once.

Synchronizations Coverage

- All synchronizations:
  - L-synchronizations



  - T-synchronizations



Synchronizations Coverage

- All-T-synchronizations
  - Cover all distinct T-synchronizations at least once.
- All-L-synchronizations
  - Cover all distinct L-synchronizations at least once.

Synchronizations Coverage

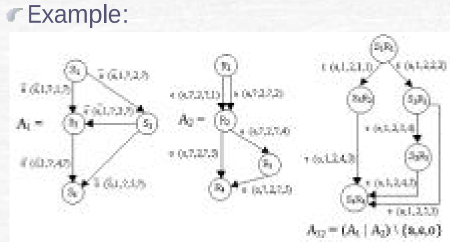
- Interface Synchronizations (bottom up incremental testing)
  - Take advantage of incremental testing.
  - Focus on detection of faults involving interface-level synchronizations, since lower-level synchronizations have already been covered by test paths.

Synchronizations Coverage

- Interface Synchronizations (continued)
  - All-int-transitions
  - All-int-T-synchronizations
  - All-int-L-synchronizations

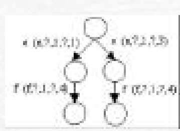
Synchronizations Coverage

- Example:



Synchronizations Coverage

- Covering All Synchronizations Incrementally:



  - Definitions of external equivalent paths are based on transition labels, not annotations.
  - Prune must consider annotations.

Critique

- Incremental approach is a nice idea:
  - Work done at lower level is passed upwards so relatively little work is needed at upper levels.
  - Reduction is a crucial thing.
- Bad things:
  - No mention of coverage criteria for program slice approach.
  - Prune should be modified before publishing.

Conclusion

- Incremental approach to testing of concurrent programs.
- Advantages:
  - Alleviates state explosion problems.
  - Supports incremental development and testing.
  - Focuses on faults in the interactions of concurrent processes.