Reachability Graphs

Analyzing Concurrent & Distributed Systems

• Dynamic analysis approaches
  • Monitor and replay
  • Coverage criteria
  • Specification-based evaluation
  • FSAs and QREs
  • Temporal Logics

• Static analysis approaches
  • Reachability graphs and reachability analysis
  • Petri nets and Petri net based analysis
  • CFGs and CFG based analysis
  • Finite State Verification
    • Model checking
    • Flow equations
    • Dataflow analysis

Reachability Graph

• models state space
  • Each node represents a possible state in a distributed system
    • States represent the value of all the variables, including the program counter for each task
    • If we only consider the value of the program counter for each task then each state is a vector where the ith element is the current program counter of the ith task
    • \(<pc_1, pc_2, ... pc_i>\)
    • Coarse-grained representation of a RG: doesn’t consider values of variables

Reachability Graph

• Typically, each edge represents progress in a single task
  • Multiple concurrent events may be possible, but allowing only single events captures all states and simplifies the graph structure (interleaved execution model)
  • Only have multiple tasks progress when required by the semantics of the programming construct
    • E.g., rendezvous
  • Only contains states that are potentially reachable from the start state

Reading Assignment


Reachability Graph Example

task control flow graphs

T1

begin

q

T2.Q

Accept Q

e1

end

T2

begin

b1

b2

q

q'

q,e2

b1.q'

b1,q'

b2.q

b2,q

e1,q'

e1,q

q,q'

e2,q

b1.b2

b1.b_q

b2.b_q

r(q,q')

b_q.b2

b_q,b2

b_q,b1

b_q,b_q'

b_q,b_q

r(q,q')

T1

T2

Accept Q

Accept Q

end

end

Worklist: <b1,b2> <b_q,b2> <b1,b_q> <r(q,q')> <e1,q'> <q,e2> <e1,e2>

Alternative: Show both tasks blocked before a rendezvous

Use a worklist to build the reachability graph

Infeasible synchronizations

begin

T2.Q

end

begin

T2.Q

end

T1

begin

T2.begin

T1.begin

begin

T2.begin

Infeasible synchronization

Infeasible synchronization?

precise up to symbolic execution

begin

T2.Q

Accept Q

T2.Q

Accept Q

end

end

T1

begin

b_2.5

b_2.6

b_3.6

b_3.7

b_4.7

b_4.8

b_5.6

b_6.7

b_7.8

(1,5)

(2,6)

(3,7)

(4,8)

(5,6)

(6,7)

(7,8)
Another example

Analysis of the reachability graph
• State based
  • Look at individual nodes
    • E.g., deadlock
  • (sub)Path based
    • Sequences of states

State based analysis
• Check some characteristic of each reachable state
  • E.g., deadlock, race conditions
  • E.g., can a reader be reading and a writer writing at the same time?
  • To solve, examine each state in graph for the characteristic

Path based analysis
• Checking characteristics of sequences of events in the reachability graph
  • Examples:
    • Must always write at least once before reading
    • Is it not possible to have a consumer consume before a producer produces?
  • To solve, we examine all paths in the graph for the sequence
    • Number of paths may be infinite
    • Can use dataflow state propagation to check path based properties
      • Creates equivalence classes of nodes with respect to the property being examined
Reachability graph based flow analysis

Shared Resource Example for a Reactive System

class Example extends Thread {
    private final int id;
    private final CyclicBarrier barrier;

    public void run() {
        while(true) {
            x = id;
            barrier.await();
            if(x == id) {
                use_resource();
            } // end while
        } // end run
    } // end class

Example Program with 2 threads

Thread 1:
    while(true) {
        x = 1;
        barrier.await();
        if(x == 1) {
            use_resource();
        }
    }

Thread 2:
    while(true) {
        x = 2;
        barrier.await();
        if(x == 2) {
            use_resource();
        }
    }

Reachability Graph

Either Thread 1 or 2 can take a step
• Suppose Thread 1 takes a step

Reachability Graph

Either Thread 1 or 2 can take a step
• Suppose Thread 2 takes a step

Reachability Graph

Example Program with 2 threads

Thread 1:
    while(true) {
        x = 1;
        barrier.await();
        if(x == 1) {
            use_resource();
        }
    }

Thread 2:
    while(true) {
        x = 2;
        barrier.await();
        if(x == 2) {
            use_resource();
        }
    }

Reachability Graph

Thread 1:
    while(true) {
        x = 1;
        barrier.await();
        if(x == 1) {
            use_resource();
        }
    }

Thread 2:
    while(true) {
        x = 2;
        barrier.await();
        if(x == 2) {
            use_resource();
        }
    }

Reachability Graph

Thread 1:
    while(true) {
        x = 1;
        barrier.await();
        if(x == 1) {
            use_resource();
        }
    }

Thread 2:
    while(true) {
        x = 2;
        barrier.await();
        if(x == 2) {
            use_resource();
        }
    }

Reachability Graph
Checking Properties

1. Freedom from deadlock
   • Are there any nodes without outgoing edges?

2. Mutual exclusion
   • Are there any nodes with both PCs on line 3?

Checking Deadlock

Thread 1:
while(true) {
  0: x = 1;
  1: barrier.await();
  2: if(x == 1)
  3: use_resource();
}

Thread 2:
while(true) {
  0: x = 2;
  1: barrier.await();
  2: if(x == 2)
  3: use_resource();
}

Checking Mutual Exclusion

Thread 1:
while(true) {
  0: x = 1;
  1: barrier.await();
  2: if(x == 1)
  3: use_resource();
}

Thread 2:
while(true) {
  0: x = 2;
  1: barrier.await();
  2: if(x == 2)
  3: use_resource();
}
Checking Properties

1. Freedom from deadlock
   - Are there any nodes without outgoing edges?

2. Mutual exclusion
   - Are there any nodes with both PCs on line 3?

3. Liveness
   - The resource will eventually be used
   - Are there any reachable cycles in which the resource is not used?

Reachability based analysis is inherently exponential

- Size of the reachability graph is exponential in the number of tasks
  - N nodes per task, T tasks
    - worse case bound on the size of the graph:
      - N^T nodes in the reachability graph

- Data flow analysis is often quadratic in the size of the graph
  - (N^T)^2
Controlling Complexity of Reachability Analysis

- Don’t consider all interleavings of events, only consider “representative” interleavings
- Valmari, Godefroid, Wolper, McDowell
- Use compositional techniques
  - Analyze reachable states of portions of the model and summarize
  - Still have exponential worst-case upper bound

Additional Problems With Reachability Analysis

- Imprecision
  - Model may not capture all information about state
  - For example, may not model all variable values
  - Aliasing may cause inclusion of events and states that are not actually possible
  - Over-approximate executable paths
    - Can lead to the consideration of infeasible paths

Non-Conservative Techniques

- Non-conservative techniques may not find a problem even though one exists
- For reachability graphs, a common non-conservative technique is to bound the size of the graph
  - Limit depth of the graph (limit length of path from start state to any node)
  - Limit number of loop iterations
- Non-conservative techniques may under-report errors
  - False negative
- Conservative techniques may over-report errors
  - False positive

Summary of Reachability Analysis

- Reachability analysis is intuitively appealing, but difficult to implement efficiently (sub-exponentially)
- Techniques exist to control state explosion, but they still carry an exponential upper bound
  - I.e., May be practical on some problems
- Looked at a CFG based reachability graph
  - Can also be constructed from Petri Net representations

Petri nets

A Petri Net is a four-tuple, \( C = (P, T, I, O) \)
- \( P = \{p_1, p_2, \ldots, p_n\} \) is a finite set of places
- \( T = \{t_1, t_2, \ldots, t_m\} \) is a finite set of transitions
- \( I: T \rightarrow P \) is the input function
  - \( p_i \) is an input place of a transition \( t_j \) if \( p_i \in I(t_j) \)
- \( O: T \rightarrow P \) is the output function
  - \( p_i \) is an output place of a transition \( t_j \) if \( p_i \in O(t_j) \)
- \( \mu_0 = \{u_{01}, u_{02}, \ldots, u_{0n}\} \) the initial marking
- \( M: P \rightarrow \) integer is the number of tokens at place \( p \)
Petri net firing rule

- A transition \( t \) is enabled if and only if \( \forall p_i \in I(t), m(p_i) > 0 \)
- Firing an enabled transition \( t \) produces a new marking \( m' \)
  - \( m'(p_i) = m(p_i)-1, \forall p_i \in I(t) \)
  - \( m'(p_i) = m(p_i)+1, \forall p_i \in O(t) \)
  - \( m'(p_i) = m(p_i) \) otherwise

Transition firing examples

Non-deterministic choice example

Synchronous task interaction

CFG representation

example

with task T0 is
  task T1 is
    entry P;
    entry Q;
end T1;
task T2;
begin
  T1.P;
end task;

Synchronous task interaction

- Initial marking has one token per task
- Task interaction preserves the number of tokens per task

11 begin
12 T1.P
13 end

1 begin
2 T1.P
3 end

1 begin
2 accept P
3 accept Q
4 T1.P
5 select
6 accept P
7 accept Q
8 T1.P
Reachability Analysis from a Petri net

- The reachability graph, $R = (N,E)$, for Petri net $= (P,T,I,O)$
  - $N = n_1, n_2, \ldots, n_l$, each $n_i$ corresponds to a Petri net marking $\mu_i = (a_{i1}, a_{i2}, \ldots, a_{in})$
  - $E = \{(\ldots, (n_i, n_j), \ldots)\}$ where there is a transition from $p_{ik}$ to $p_{ir}$ and a series of markings that cause that transition to be taken.
- reachability graph for a safe conservative Petri net with $n$ places and $k$ tokens can potentially have $n^k$ nodes
  - thus, reachability graphs for Petri nets are potentially exponential in the number of tasks in the program.

Petri net reachability graph analysis

- State properties
  - checking some characteristic of each reachable state
    - examples: deadlock, critical races, multiple readers
- Path properties
  - Check for sequences of events
  - Basically doesn't matter if the reachability graph is derived from control flow graph or from Petri net models
deadlock detection

Problems with reachability analysis
- complexity
  - many reachability problems shown to be NP-complete
  - upper bound on graph size is \((\text{average task size}) \times (\text{number of tasks})\)
  - commonly called the "state explosion problem"
- 10 tasks, 10 states in each --> 10 billion states
- some contributors
  - consideration of all possible interleavings of events
  - nondeterminism
- reachability analysis still has the static analysis problems of imprecision
  - alias resolution
  - path infeasibility

Benefits of Reachability Analysis
- Can often be optimized to produce interesting results
  - E.g., SPIN, G. Holzmann, AT&T
  - Works with a "simplified specification language," Promela
  - State Transition Model
  - Publicly available tool
- Don't have to analyze the whole system
  - Can evaluate subsystems
  - Can evaluate specific configurations
  - Can evaluate high-level designs
  - Better to find a problem earlier than later
- For concurrent/distributed systems, provides some assurances