Chapter 4

Graph theory for Testers

Linear Graph Theory

- A branch of topology—focus on connections
- (undirected) Graphs
  - nodes, edges, matrices
  - degree of a node
  - Paths
  - Components
- Directed graphs
  - nodes, edges
  - indegree, outdegree
  - paths and semi-paths
  - n-connectedness
  - cyclomatic number
  - strong and weak components
  - Program Graphs

Biblical advice…

“Test everything; keep that which is good”

1 Thessalonians 5:21

[Graph theory will help us understand the “everything” part.]

Linear Graphs

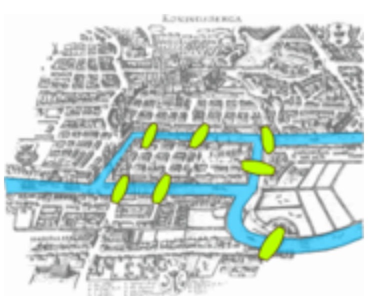
Definition 1: A *graph G = (V, E)* is composed of a finite (and nonempty) set V of nodes and a set E of unordered pairs of nodes.

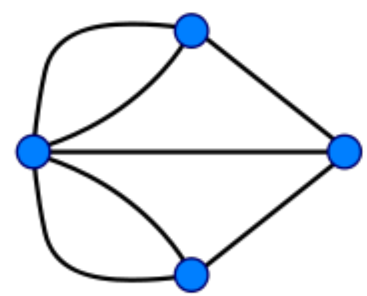
- When drawn, graphs usually show nodes as circles, and edges as lines.
- Leonhard Euler developed graphs as a way to solve the "Bridges of Königsberg" puzzle in 1735.
- (Königsberg, Prussia is now Kaliningrad, Russia)

Bridges of Königsberg Puzzle

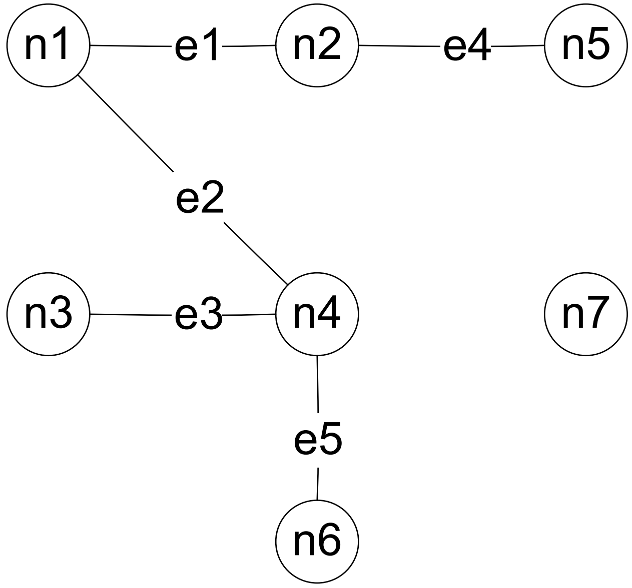
(http://en.wikipedia.org/wiki/Seven_Bridges_of_Köninigsberg)

- Take a walk around the city, traversing each bridge exactly once.
- Euler's formulation made the problem easier to solve.
- (It is impossible)
- Nice example of using math to solve an abstract problem.





Linear Graph (running example)



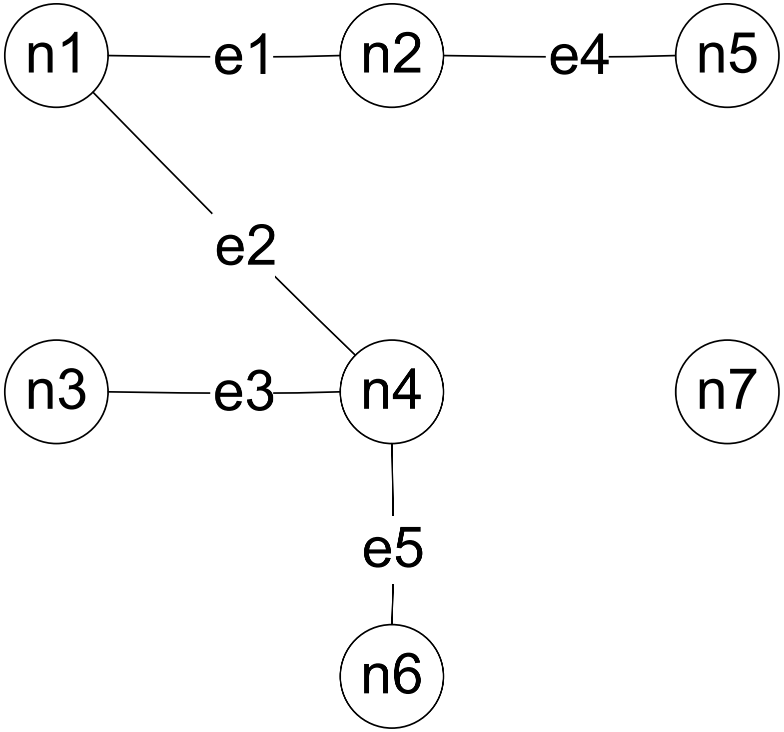
V = {n1, n2, n3, n4, n5, n6, n7}
E = {e1, e2, e3, e4, e5}
= {(n1, n2), (n1, n4), (n3, n4), (n2, n5), (n4, n6)}

Degree of a Node

Definition 2: The *degree of a node* in a graph is the number of edges that have that node as an endpoint.

- We write deg(n) for the degree of node n.
- "popularity" of a node in a graph
- Social scientists use graphs to describe
  - social interactions
  - friendship
  - communication

Degree of a Node



deg(n1) = 2
deg(n2) = 2
deg(n3) = 1
deg(n4) = 3
deg(n5) = 1
deg(n6) = 1
deg(n7) = 0

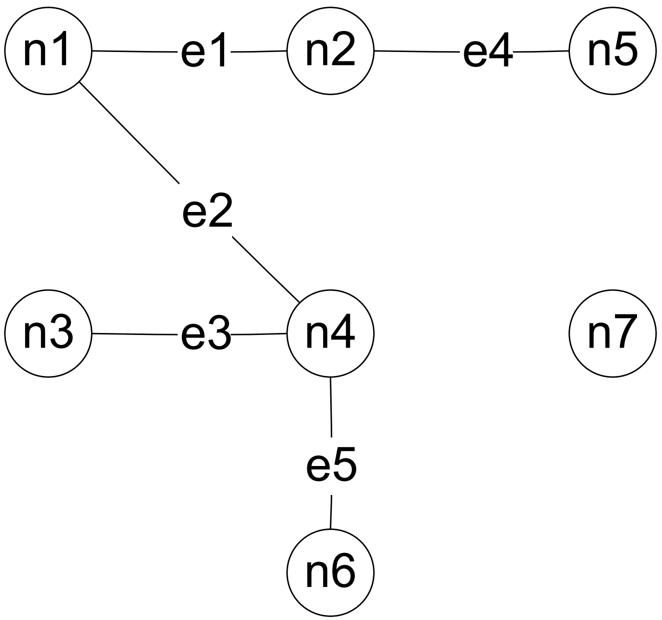
Incidence Matrix of a Graph

Definition 3: The *incidence matrix of a graph* G = (V, E) with m nodes and n edges is an m by n matrix, where the element in row i, column j is a 1 if and only if node i is an endpoint of edge j; otherwise, the element is 0.

Sample Incidence Matrix

(What must be true about columns in an incidence matrix?)

	e1	*e2*	*e3*	*e4*	*e5*
n1	1	1	0	0	0
n2	1	0	0	1	0
n3	0	0	1	0	0
n4	0	1	1	0	1
n5	0	0	0	1	0
n6	0	0	0	0	1
n7	0	0	0	0	0



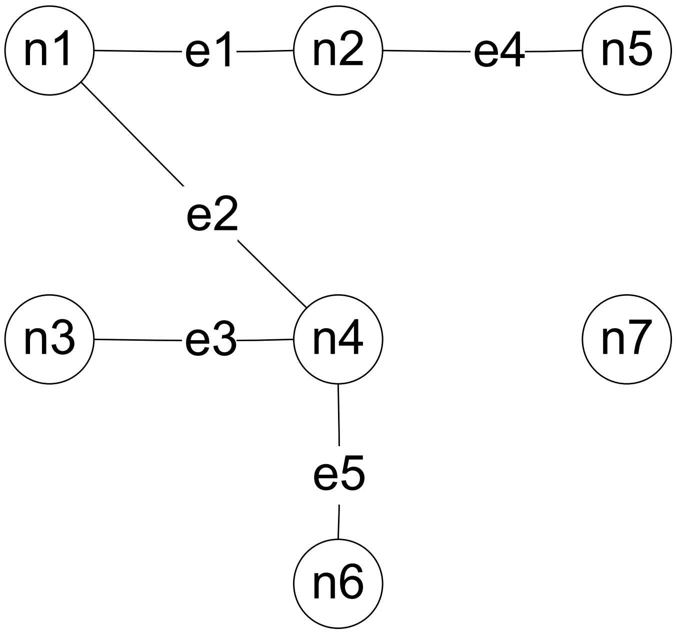
Adjacency Matrix of a Graph

Definition 4: The *adjacency matrix of a graph* G = (V, E) with m nodes is an m by m matrix, where the element in row i, column j is a 1 if and only if an edge exists between node i and node j; otherwise, the element is 0.

- This matrix can be used to answer questions about a program...
  - reachability
  - points affected by a fault
  - connectedness in a program
- Question: Why is this adjacency matrix symmetric?

Sample Adjacency Matrix

	*n*1	*n*2	*n*3	*n*4	*n*5	*n*6	*n*7
n1	0	1	0	1	0	0	0
n2	1	0	0	0	1	0	0
n3	0	0	0	1	0	0	0
n4	1	0	1	0	0	1	0
n5	0	1	0	0	0	0	0
n6	0	0	0	1	0	0	0
n7	0	0	0	0	0	0	0



Paths in a Graph

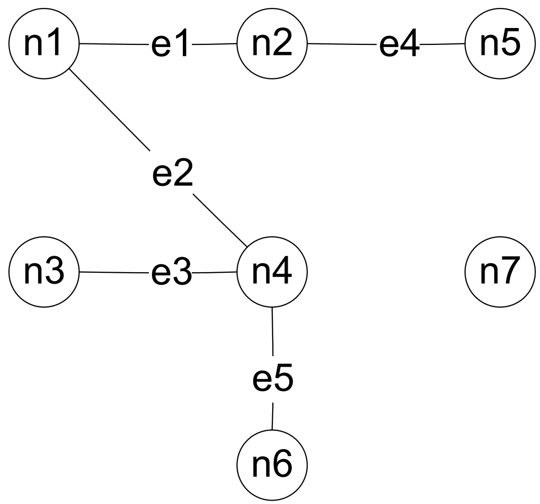
Definition 5: A *path* is a sequence of edges such that, for any adjacent pair of edges ei, ej in the sequence, the edges share a common (node) endpoint.

- Paths capture/express connectivity.
- Paths can be described by
  - sequences of nodes, or
  - sequences of edges

Some Paths

path between

– n1 and n5	n1, n2, n5	e1, e4
– n6 and n5	n6, n4, n1, n2, n5	e5, e2, e1, e4
– n3 and n2	n3, n4, n1, n2	e3, e2, e1



Connectedness

Definition 5: Two *nodes are connected* if and only if they are in the same path.

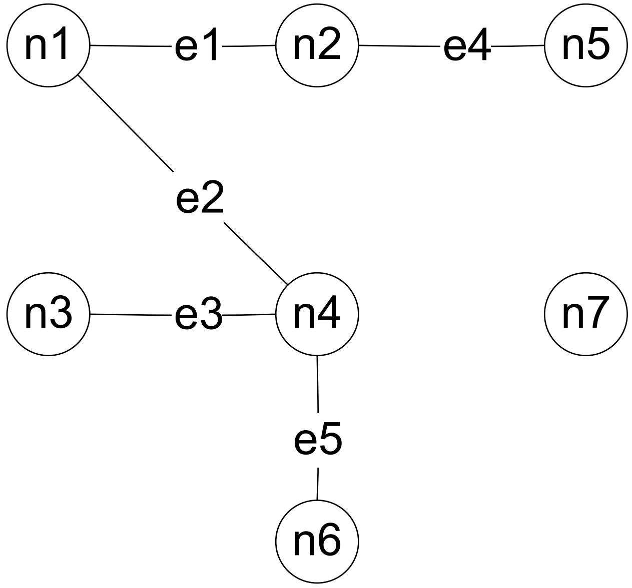
- Question: Is connectedness in a graph an equivalence relation?
- From Chapter 3, to be an equivalence relation, connectedness must be
  - reflexive
  - symmetric
  - transitive

Components of a Graph

Definition 6: A *component of a graph* is a maximal set of connected nodes.

- Question: Does connectedness help define components?
- Components are "maximal"

Components of "our" Graph



C1 = {n1, n2, n3, n4, n5, n6}
C2 = {n7}

Condensation Graphs

Definition 7: Given The *condensation graph* of a graph G = (V, E) is formed by replacing each component of G by a condensing node.

Questions:

- Draw the condensation graph of our running example.
- Can there be any edges in a condensation graph?

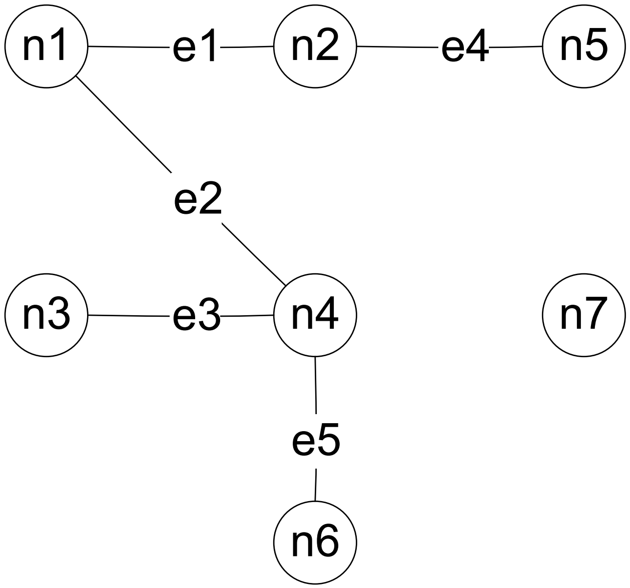
Cyclomatic Number of a Graph

Definition 8: The *cyclomatic number of a graph* G is given by $V(G) = e - n + p$, where

- e is the number of edges in G
- n is the number of nodes in G
- p is the number of components in G

- Cyclomatic complexity pertains to both ordinary and directed graphs (next topic).
- V(G) is sometimes called McCabe Complexity after Thomas McCabe.
- (not very useful on our running example)

Cyclomatic Complexity of "our" Graph



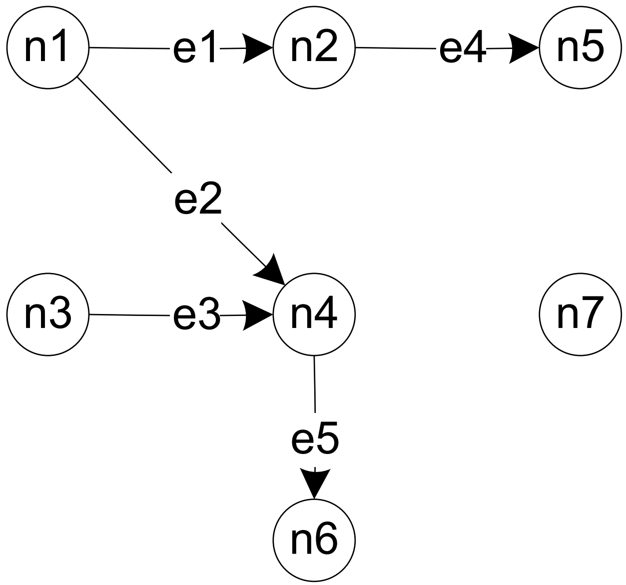
$$V(G) = e - n + p$$
$$= 5 - 7 + 2$$
$$= 0$$

Directed Graphs

Definition 9: A *directed graph (or digraph) D = (V, E)* consists of a finite set V of nodes, and a set E of edges, where each edge ek = <ni, nj> is an ordered pair of nodes ni, nj ϵ V.

- <x, y> is an ordered pair
- (x, y) is an unordered pair
- For an edge <a, b> in a directed graph, node a is the initial node, and node be is the terminal node..

Directed Graph (revised example)



V = {n1, n2, n3, n4, n5, n6, n7}
E = {e1, e2, e3, e4, e5}
= {<n1, n2>, <n1, n4>, <n3, n4>, <n2, n5>, <n4, n6>}

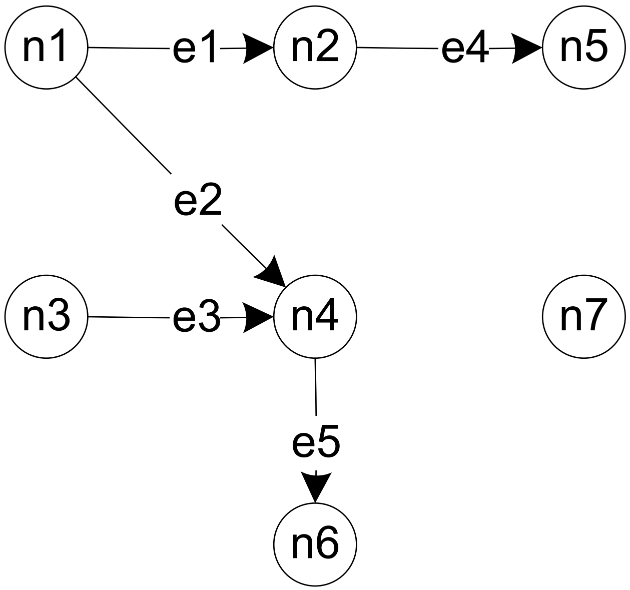
Indegrees and Outdegrees of a Node

- Definition 10: The *indegree of a node* in a directed graph is the number of distinct edges that have the node as a terminal node.
- Definition 11: The *outdegree of a node* in a directed graph is the number of distinct edges that have the node as a start point.

- We write indeg(n) and outdeg(n).

Indegrees and Outdegrees

indeg(n1) = 0 outdeg(n1) = 2
indeg(n2) = 1 outdeg(n2) = 1
indeg(n3) = 0 outdeg(n3) = 1
indeg(n4) = 2 outdeg(n4) = 1
indeg(n5) = 1 outdeg(n5) = 0
indeg(n6) = 1 outdeg(n6) = 0
indeg(n7) = 0 outdeg(n7) = 0



Types of Nodes

Definition 12:

A node with indegree = 0 is a *source node*.

A node with outdegree = 0 is a *sink node*.

A node with indegree ≠ 0 and outdegree ≠ 0 is a *transfer node*

- Question: What are the source, sink, and transfer nodes in our continuing example?

Adjacency Matrix

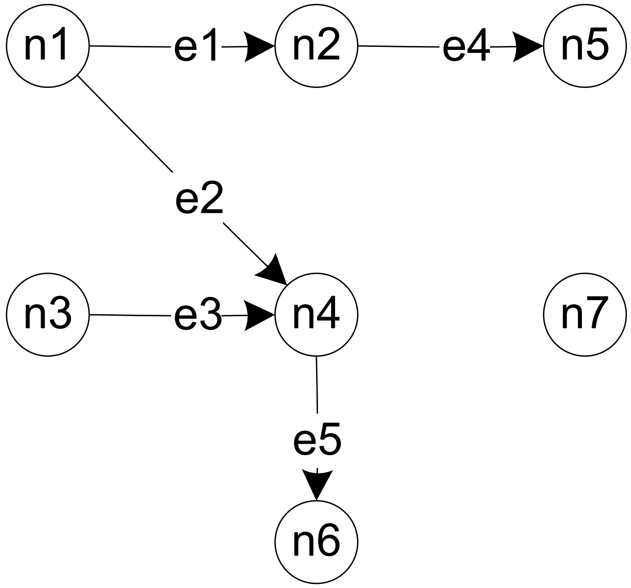
Definition 13: The *adjacency matrix of a directed graph* D = (V, E) with m nodes is an m by m matrix

A = (a(i, j)) where a(i, j) is a 1 if and only if there is an edge from node i to node j; otherwise, the element is 0

- Only in very special graphs are the adjacency matrices symmetric.

Adjacency Matrix

	*n*1	*n*2	*n*3	*n*4	*n*5	*n*6	*n*7
n1	0	1	0	1	0	0	0
n2	0	0	0	0	1	0	0
n3	0	0	0	1	0	0	0
n4	0	0	0	0	0	1	0
n5	0	0	0	0	0	0	0
n6	0	0	0	0	0	0	0
n7	0	0	0	0	0	0	0



Adjacency Matrix Questions

1. Can indegrees and outdegrees of a node be derived (observed) in an adjacency matrix of a directed graph?
2. Can types of nodes (source, sink, transfer) be derived (observed) in an adjacency matrix of a directed graph?
3. Can paths be derived (observed) in an adjacency matrix of a directed graph? Hint: think about powers of an adjacency matrix.

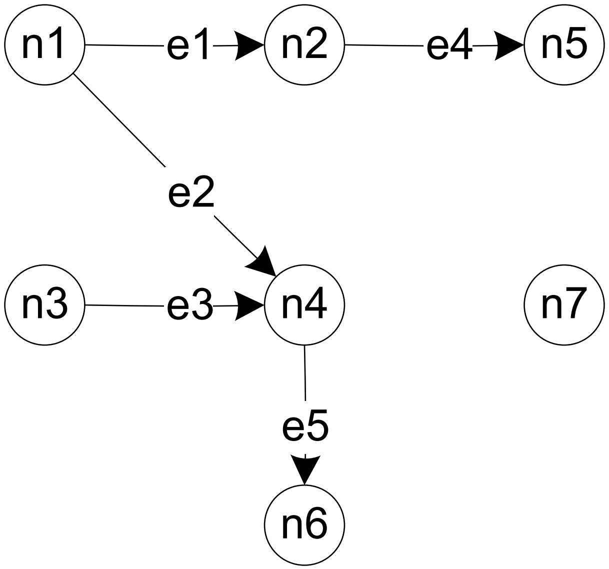
Paths in a Directed Graph

Definition 14: A *(directed) path* is a sequence of edges such that, for any adjacent pair of edges ei, ej, in the sequence, the terminal node of the first edge is the initial node of the second edge.

Some related definitions...

- A *cycle* is a path in which some node is both the initial and the final node in the path.
- A chain is a sequence of nodes in which every interior node has indegree = outdegree = 1
- A *semipath* is a sequence of edges such that at least one node is either the initial node or the terminal node of two adjacent edges in the sequence.

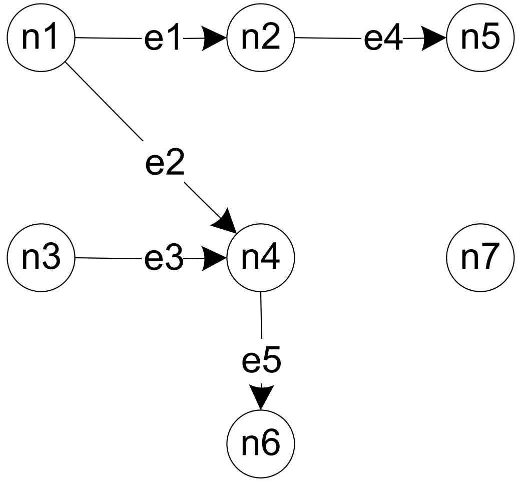
Directed Graph (new example)



There is a path from n1 to n6, and there are semipaths between nodes n1 and n3, n2 and n4, and between nodes n5 and n6.

Questions

1. List the chains in our continuing example.
2. What is/are the longest path(s)?
3. Are there any cycles?
4. How would you describe the difference between semipaths <n3, n4, n5> and <n4, n1, n2>?
5. Is <n6, n4, n1, n2, n5> a semipath?



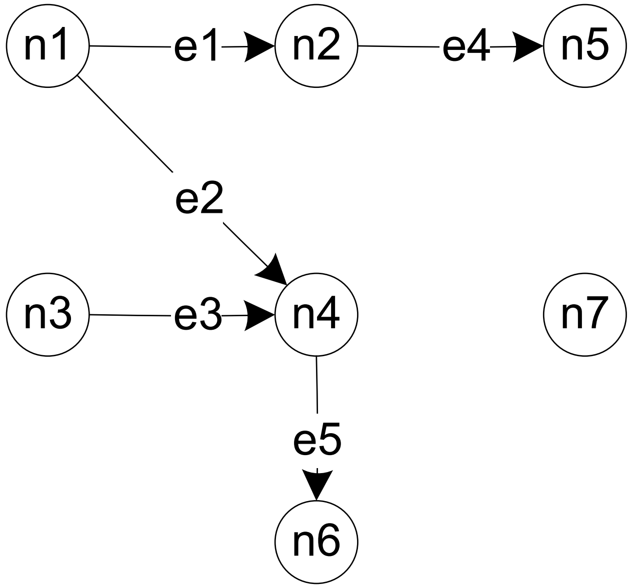
Reachability Matrix

Definition 15: The *reachability matrix of a directed graph* D = (V, E) with m nodes is an m by m matrix R = (r(i, j)), where r(i, j) is a 1 if and only if there is a path from node i to node j, otherwise the element is 0.

- Reachability is closely related to paths.
- The reachability matrix R can be computed using the adjacency matrix A of the directed graph:
  - $R = I + A + A^2 + A^3 + \ldots + A^k$
  - where k is the length of the longest path in D,
  - I is the identity matrix, and
  - powers of A are computed by slightly changed matrix multiplication in which 1 + 1 = 1

Reachability Matrix

	n1	*n2*	*n3*	*n4*	*n5*	*n6*	*n7*
n1	1	1	0	1	1	1	0
n2	0	1	0	0	1	0	0
n3	0	0	1	1	0	1	0
n4	0	0	0	1	0	1	0
n5	0	0	0	0	1	0	0
n6	0	0	0	0	0	1	0
n7	0	0	0	0	0	0	1



Compute A^2

	*n*1	*n*2	*n*3	*n*4	*n*5	*n*6	*n*7
n1	0	1	0	1	0	0	0
n2	0	0	0	0	1	0	0
n3	0	0	0	1	0	0	0
n4	0	0	0	0	0	1	0
n5	0	0	0	0	0	0	0
n6	0	0	0	0	0	0	0
n7	0	0	0	0	0	0	0

	*n*1	*n*2	*n*3	*n*4	*n*5	*n*6	*n*7
n1	0	1	0	1	0	0	0
n2	0	0	0	0	1	0	0
n3	0	0	0	1	0	0	0
n4	0	0	0	0	0	1	0
n5	0	0	0	0	0	0	0
n6	0	0	0	0	0	0	0
n7	0	0	0	0	0	0	0

	*n*1	*n*2	*n*3	*n*4	*n*5	*n*6	*n*7
n1							
n2							
n3							
n4							
n5							
n6							
n7							

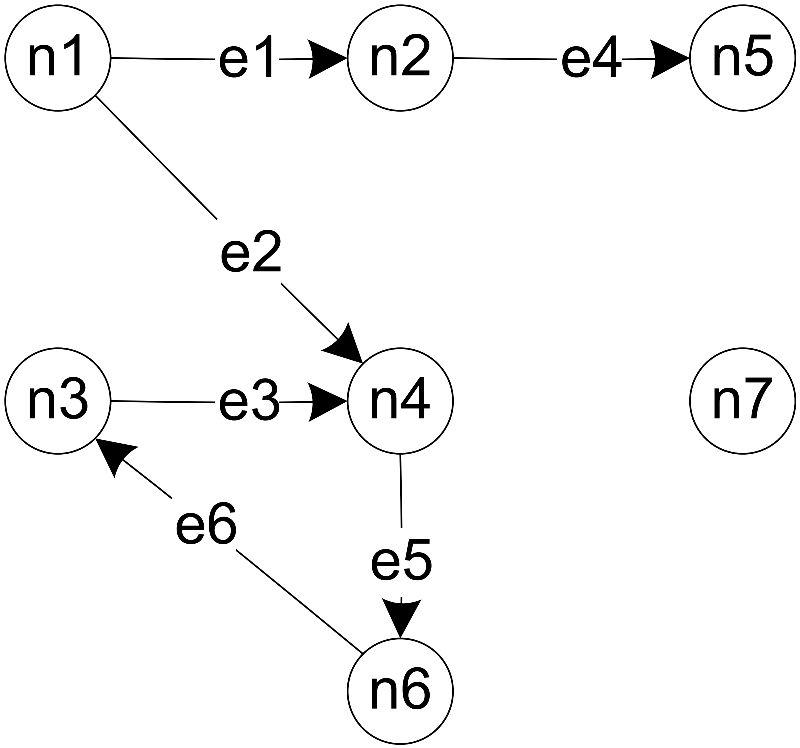
n-Connectedness

Definition 16: Nodes n_j and n_k in a directed graph are

- *0-connected* iff no path (or semipath) exists between n_j and n_k
- *1-connected* iff a semi-path but no path exists between n_j and n_k
- *2-connected* iff a path exists from n_j and n_k
- *3-connected* iff a path goes from n_j to n_k and a path goes from n to n_k

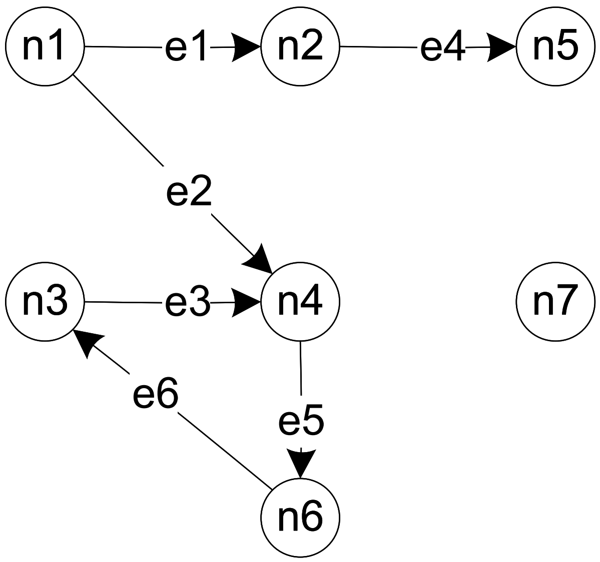
- No other degrees of n-connectedness exist
- A new edge, e6, is added to our continuing example.
- n-connectedness has very useful expressive power.

Directed Graph (third version)



Is 3-connectedness an equivalence relation?

Directed Graph (third version)



1.Find examples of a chain, a cycle, a set of 3-connected nodes

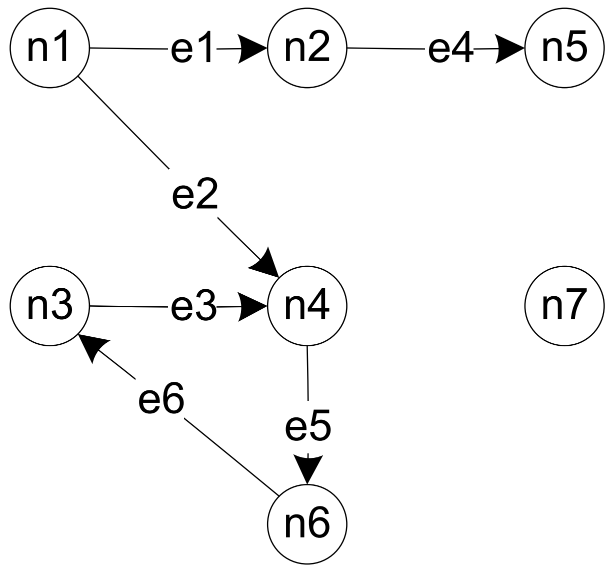
Strong Components of a Directed Graph

Definition 17: A *strong component of a directed graph* is a maximal set of 3-connected nodes.

Strong components...

- identify loops and isolated nodes.
- lead to another form of condensation graph
- support an excellent voew of testing programs with loops

Directed Graph (third version)



Strong components: S1 = {n3, n4, n6}, S2 = {n7}

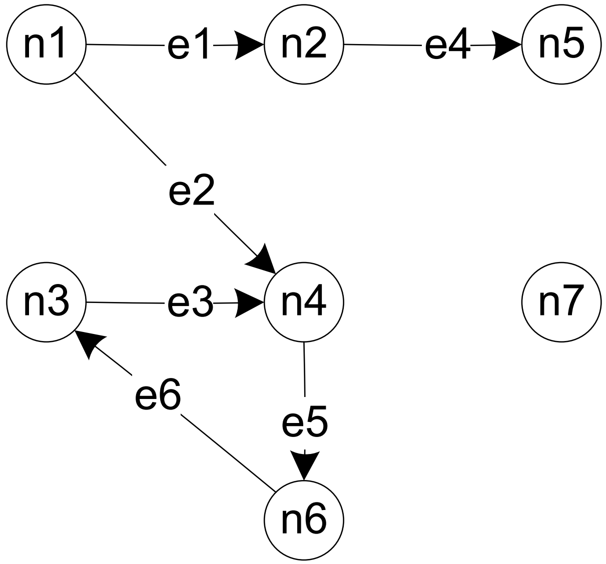
Condensation Graph of a Directed Graph

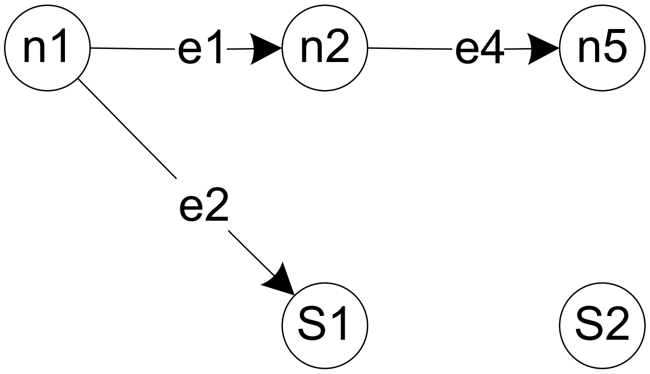
Definition 18: Given a directed graph D = (V, E), its *condensation graph* is formed by replacing strongly connected nodes by their corresponding strong components.

A condensation graph of a directed graph...

- contains no loops, and is therefore
- an Directed Acyclic Graph (DAG)
- support an excellent view of testing programs with loops

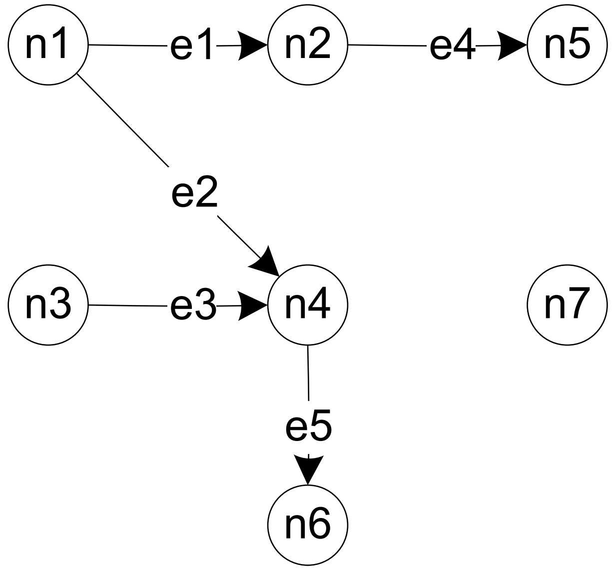
Condensation Graph of Directed Graph (third version)





Strong components: S1 = {n3, n4, n6}, S2 = {n7}

Directed Graph (new example)



V = {n1, n2, n3, n4, n5, n6, n7}
E = {e1, e2, e3, e4, e5}
= {(n1, n2), (n1, n4), (n3, n4), (n2, n5), (n4, n6)}

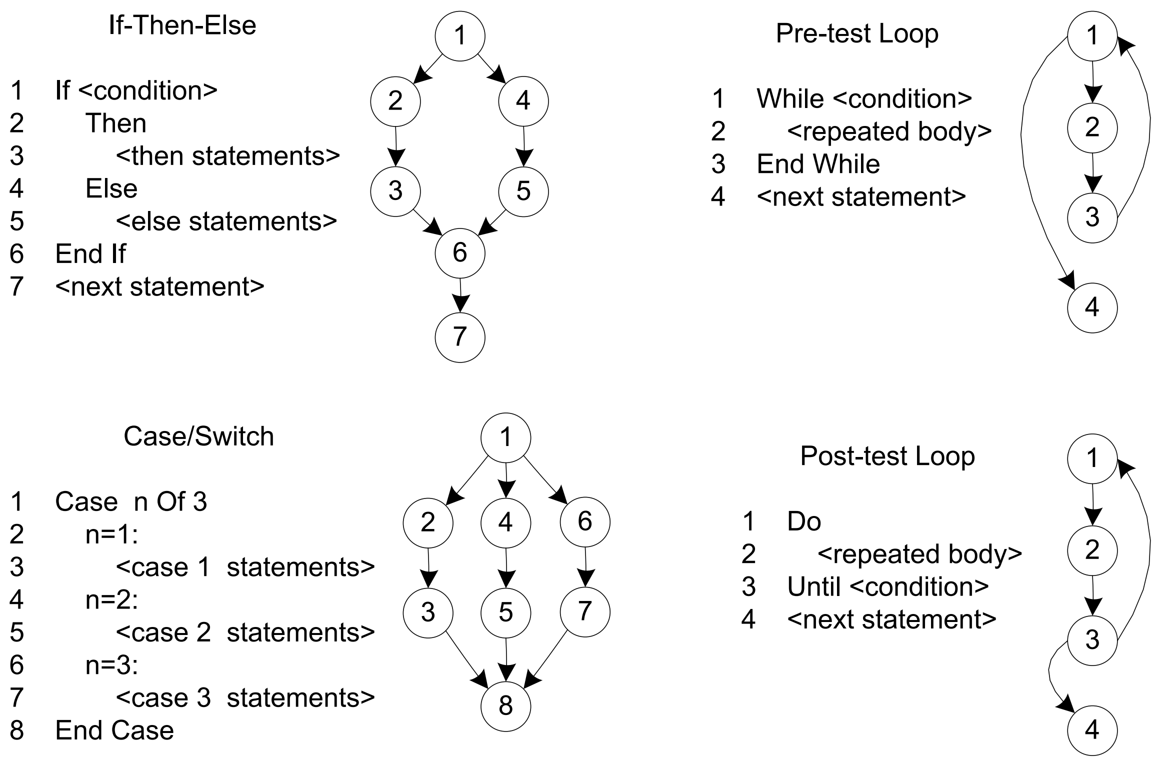
Program Graphs

Definition 19: The *program graph* of a program written in an imperative programming language is a directed graph in which nodes are either entire statements or statement fragments. There is an edge from node i to node j iff node j can be executed immediately after node i).

- (The original definitions referred to nodes as entire statements, but this doesn't fit well with modern programming languages.)
- We shall use "statement fragment" to refer either to full statements or to statement fragments.

Program Graphs

- When drawing a program graph, it is usually simpler to number the statement fragments.
- This Figure 8.1



Four Graph-Based Models

- Finite State Machines
- Petri Nets
- Event-Driven Petri Nets
- StateCharts

- These are all executable models, *i.e.,* it is possible to build a program (an engine) to execute the model.

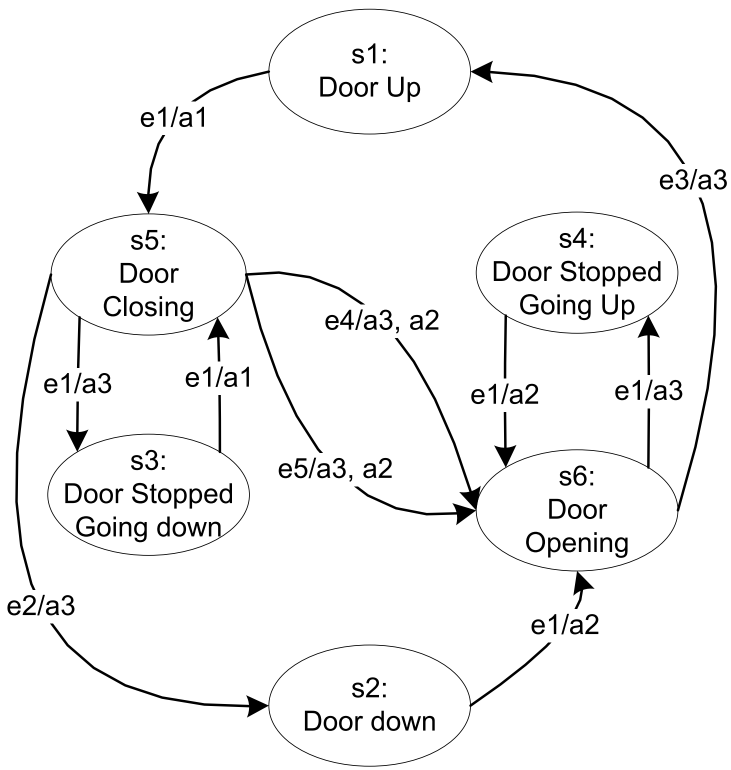
Finite State Machines (FSMs)

- Finite State Machines are directed graphs in which nodes are states, and edges are transitions from one state to a successor state.
- Transitions are caused by
  - events
  - date conditions
  - passage of time (an event)
- Constraints
  - states are mutually exclusive
  - only one transition can occur at a time
- FSMs are ideally suited for menu-driven applications

Garage Door Controller FSM

Input events
e1: depress controller button
e2: end of down track hit
e3: end of up track hit
e4: obstacle hit
e5: laser beam crossed

Output events (actions)
a1: start drive motor down
a2: start drive motor up
a3: stop drive motor
a4. door stops part way
a5. door continues opening
a6. door continues closing



Petri Nets

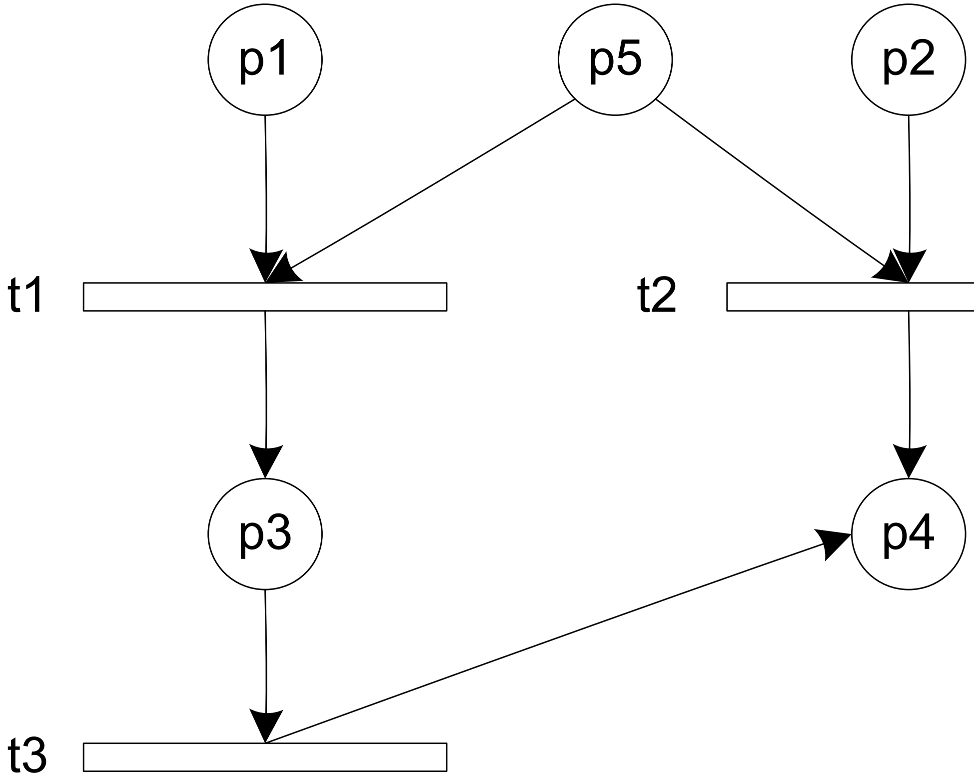
- Petri Nets are bipartite directed graphs (P, T, In, Out), where
  - P is a set of places
  - T is a set of transitions
  - In is a mapping of places to transitions, understood as input places
  - Out is a mapping of to transitions to places , understood as output places

- Finite State Machines are a special case of Petri Nets.

Petri Nets (continued)

- Petri Net execution is governed by
  - place markings
  - transition enabling and firing
  - several strategies for transition firing sequences
- Constraints
  - only one transition at a time can be fired
  - no simultaneous events
- Petri Nets can express a variety of conmplex situations.

Petri Net Example



Petri Net Exercise

Identify the following for the sample Petri Net

- the set P of places
- the set T of transitions
- the mapping In of inputs to transitions
- the mapping out of outputs of transitions

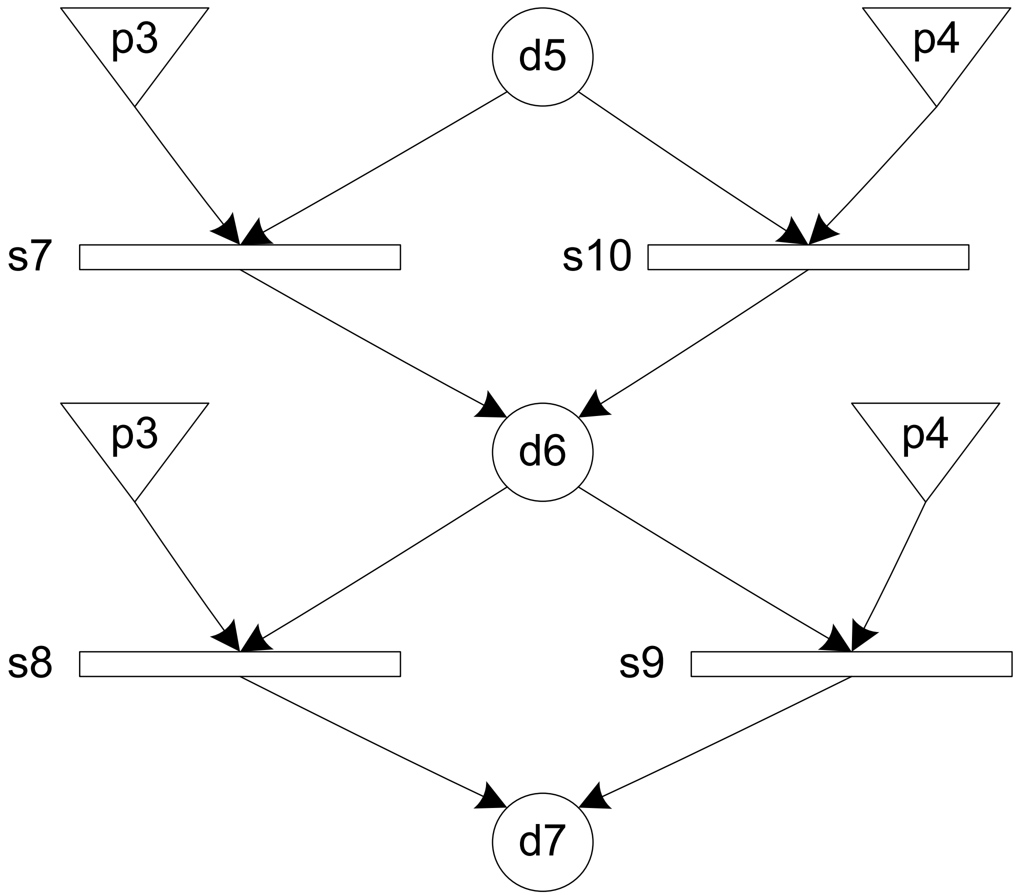
Event-Driven Petri Nets (EDPNs)

- Definition 20: An *Event-Driven Petri Net* is a tripartite-directed graph (P, D, S, In, Out) composed of three sets of nodes, P, D, and S, and two mappings, In and Out, where:
  - P is a set of port events
  - D is a set of data places
  - S is a set of transitions
  - In is a set of ordered pairs from $(P \cup D) \times S$
  - Out is a set of ordered pairs from $S \times (P \cup D)$

Event-Driven Petri Nets (continued)

- Port events can be
  - input events
  - output events
  - drawn as triangles
- Places are as in ordinary Petri Nets
- Transition firing is a simple extension of ordinary Petri Net transition firing
- (there are no output events in the example on the next slide)

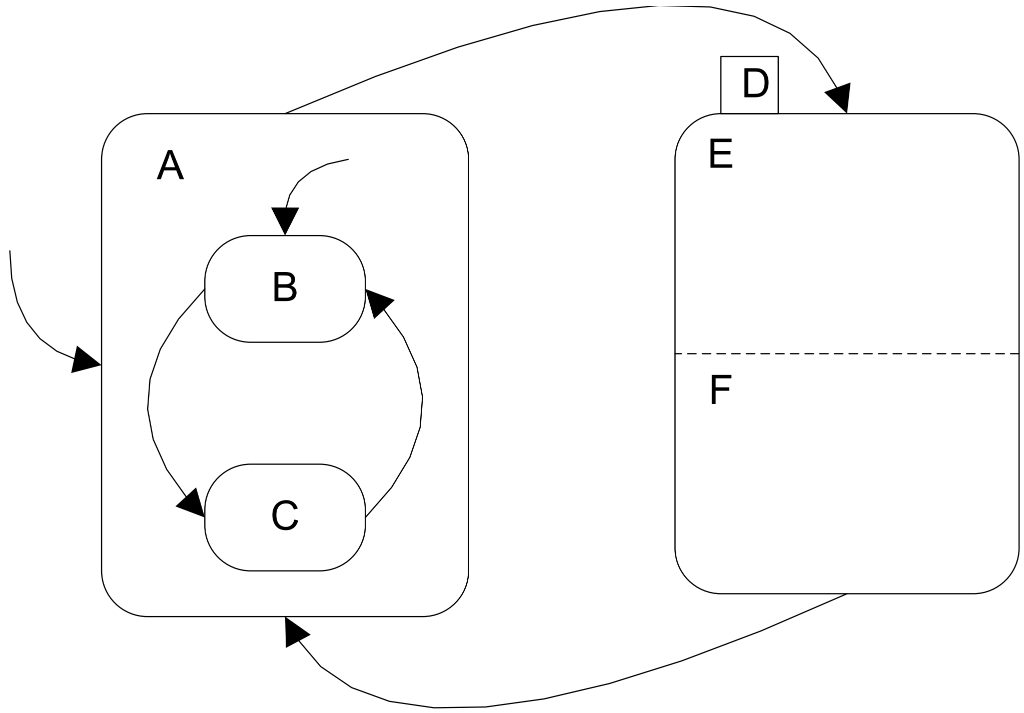
EDPN Example



StateCharts

- Created by David Harel
- Harel's goal was to combine the visual expressive power of
  - directed graphs
  - Venn diagrams
- "States" can contain...
  - lower level states
  - concurrent regions
- Transitions are very elaborate (see text)
- StateCharts are an elegant resolution to the "Finite State Machine Explosion"

Sample StateChart



Blob (state) A contains two lower level states.
Blob B contains two concurrent regions, E and F.

Wrap-Up

- Graph theory is a powerful way to express many of the concepts of software testing.
- It will be used extensively inh sequel chapters.
- The graph-based models all lead to (and support) model-based testing.
- Executable models support nearly automatic generation (derivation) os system level test cases.