Testgeneration for the FSML-Implementation Input-Generation with Coverage Criteria

Marcel Heinz

26. Februar 2014

Coverage Criteria vs Random Test Generation

1. Complete testing of all behaviors of a reactive system is impossible"
2. Solution 1 : Generating random sequences of Input-Data (Black Box)
3. Solution 2 : Generating Input-Data sequences using coverage criteria (White Box)

1

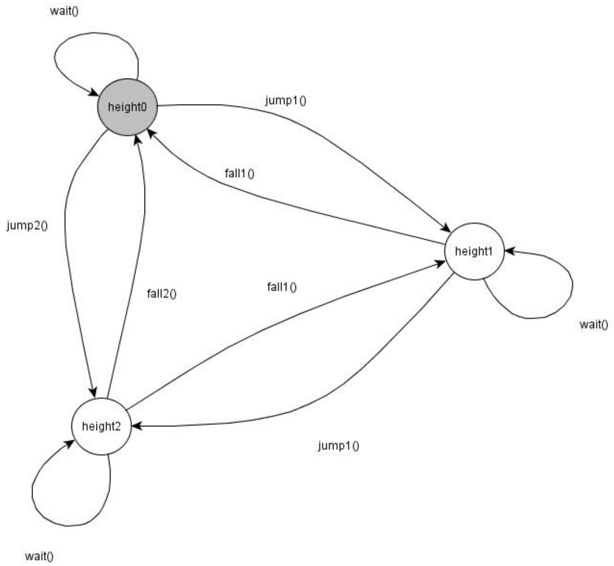
[1] See "Model-Based Testing for Embedded Systems" as a reference

Input and Flow Coverage Criteria

1. All-States
2. All-Transitions
3. ~~All-Events~~
4. Depth-n : Each run of length n from the initial state is considered in a test case.
5. All-n-Transitions: Each run of length n from any state is considered in a test case.
6. All-Paths: All possible transition sequences have to be included. $\rightarrow$ infeasible.

$\Rightarrow$ Goal: stress testing, validating execution

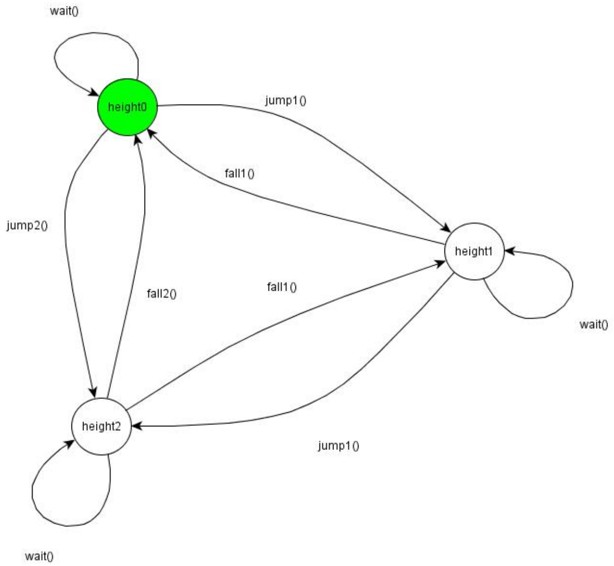
Introduction to the example FSM



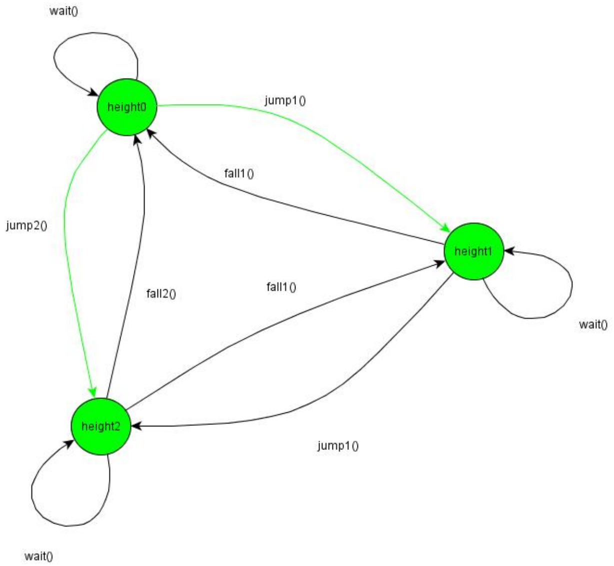
All-States Coverage

1. Bases on Breadth-First-Search
2. Starts at the initial State
3. Generates lists of Input-Sequences until every <u>State</u> has been visited at least once.

All-States Coverage Situation(I)



All-States Coverage Situation (II)



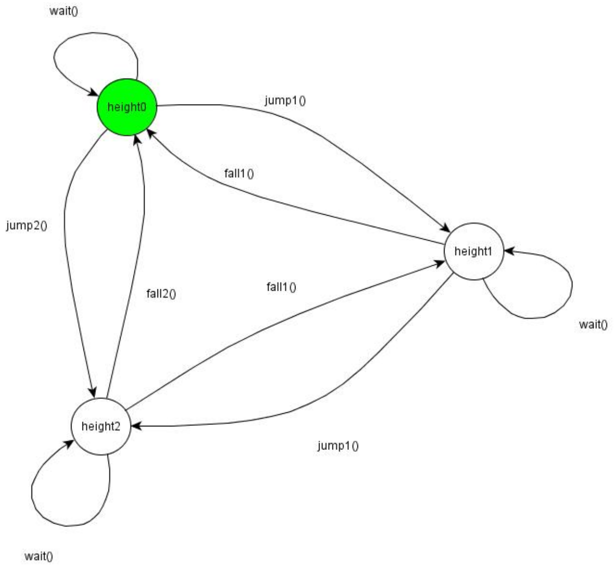
Input Lists

1. jump1()
2. jump2()

All-Transitions Coverage

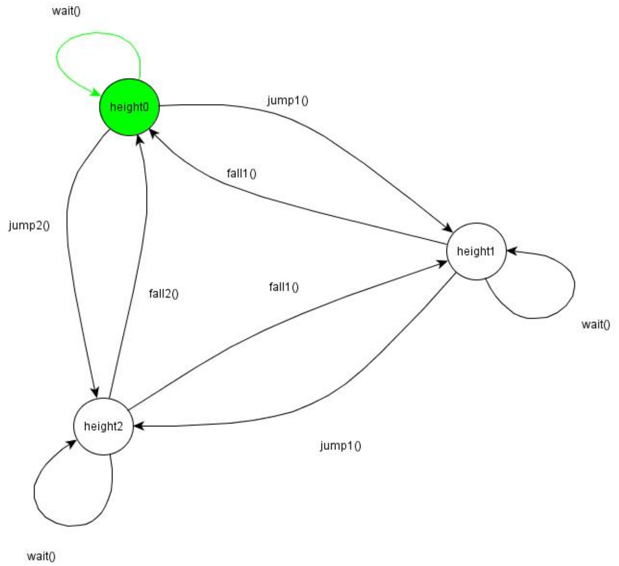
1. Bases on Breadth-First-Search
2. Starts at the initial State
3. Generates lists of Input-Sequences until every <u>Transition</u> has been visited at least once.

All-Transitions Coverage Situation (I)



Input Lists

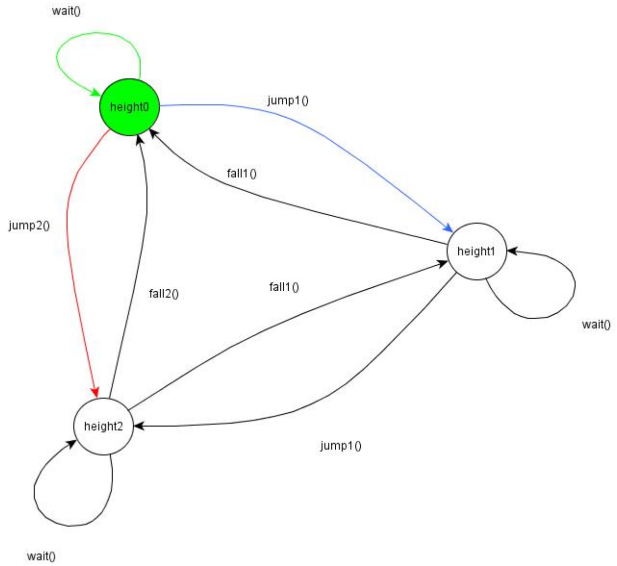
All-Transitions Coverage Situation (II)



Input Lists

1. wait()

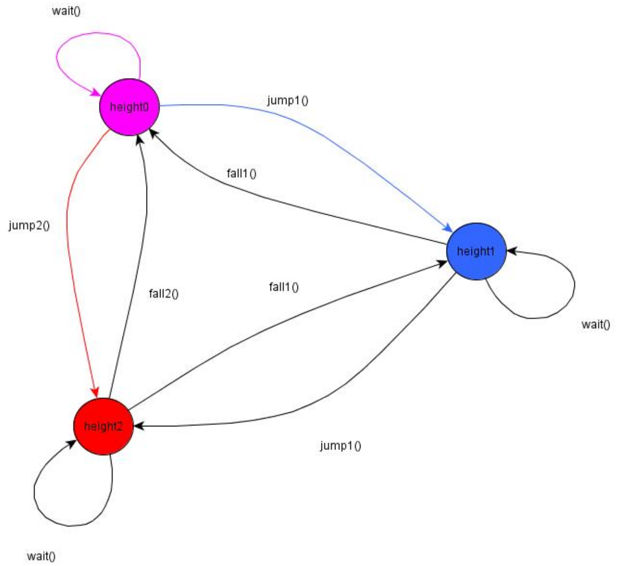
All-Transitions Coverage Situation (III)



Input Lists

1. wait(), jump1()
2. wait(), jump2()

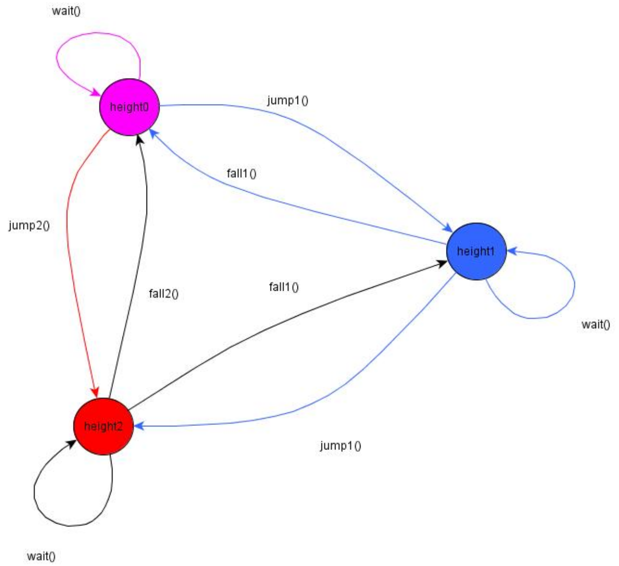
All-Transitions Coverage Situation (IV)



Input Lists

1. wait(), jump1()
2. wait(), jump2()

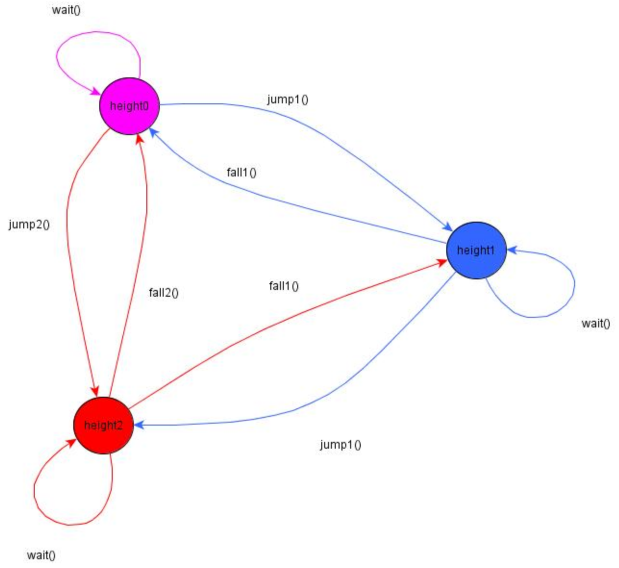
All-Transitions Coverage Situation (V)



Input Lists

1. wait(), jump1(), fall1()
2. wait(), jump1(), wait()
3. wait(), jump1(), jump1()
4. wait(), jump2()

All-Transitions Coverage Situation (VI)



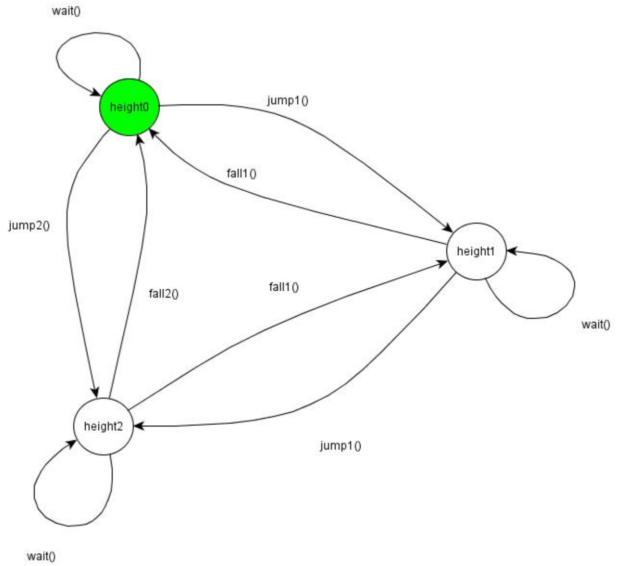
Input Lists

1. wait(), jump1(), fall1()
2. wait(), jump1(), wait()
3. wait(), jump1(), jump1()
4. wait(), jump2(), wait()
5. wait(), jump2(), fall2()
6. wait(), jump2(), fall1()

DepthN Coverage

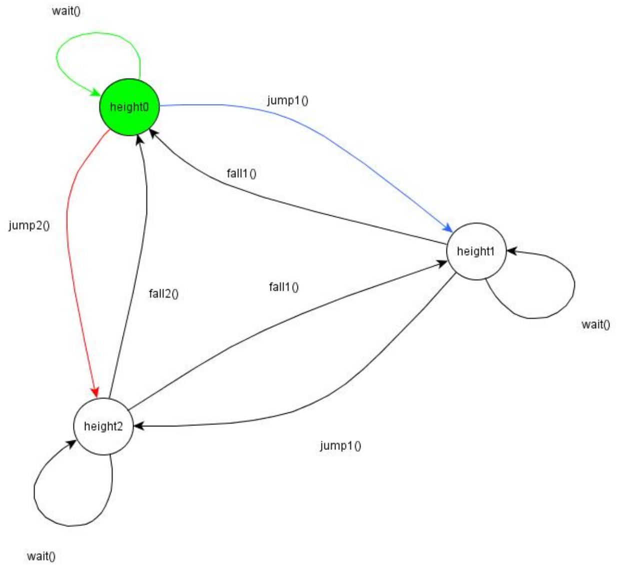
1. Bases on Breadth-First-Search
2. Starts at the initial State
3. Generates lists of all possible Input-Sequences, which have a given length.
4. For the sample we choose N = 3

DepthN Coverage Situation (I)



Input Lists

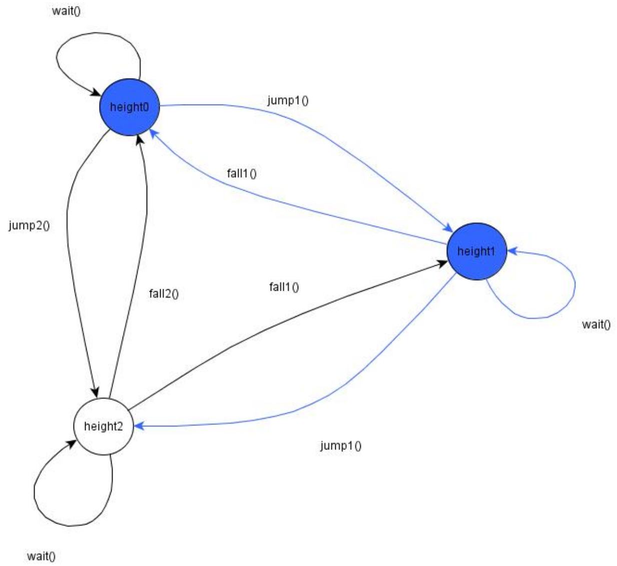
DepthN Coverage Situation (II)



Input Lists

1. wait()
2. jump1()
3. jump2()

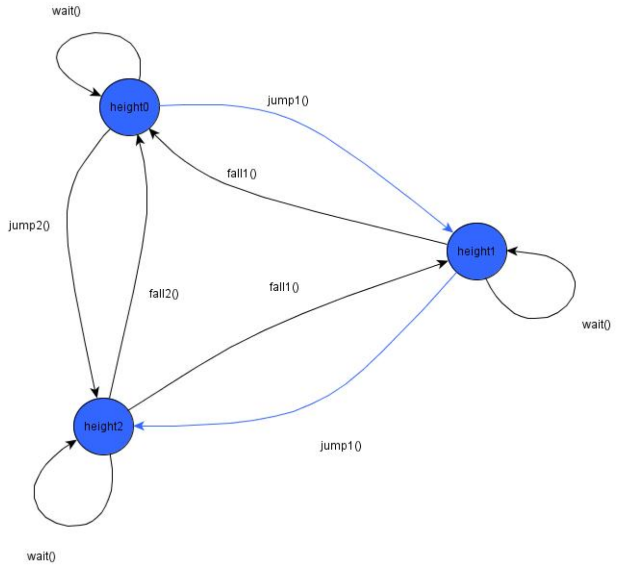
DepthN Coverage Situation (III)



Input Lists

1. wait(), wait()
2. wait(), jump1()
3. wait(), jump2()
4. jump1(), wait()
5. jump1(), fall1()
6. jump1(), jump1()
7. jump2(), wait()
8. jump2(), fall1()
9. jump2(), fall2()

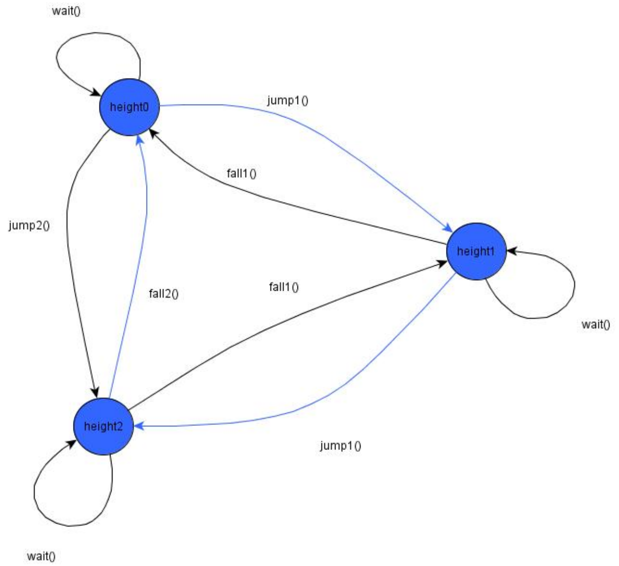
DepthN Coverage Situation (IV)



Input Lists

1. jump1(), wait()
2. jump1(), fall1()
3. jump1(), jump1()
4. ...

DepthN Coverage Situation (V)



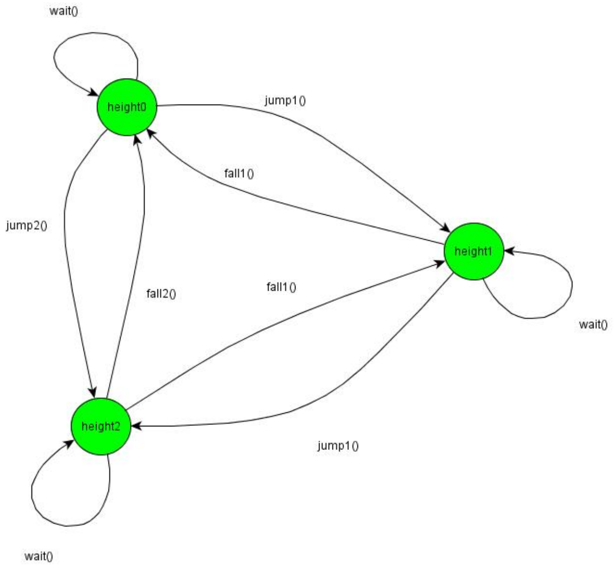
Input Lists

1. jump1(), jump1(), fall2()
2. ...

All-N-Transitions Coverage

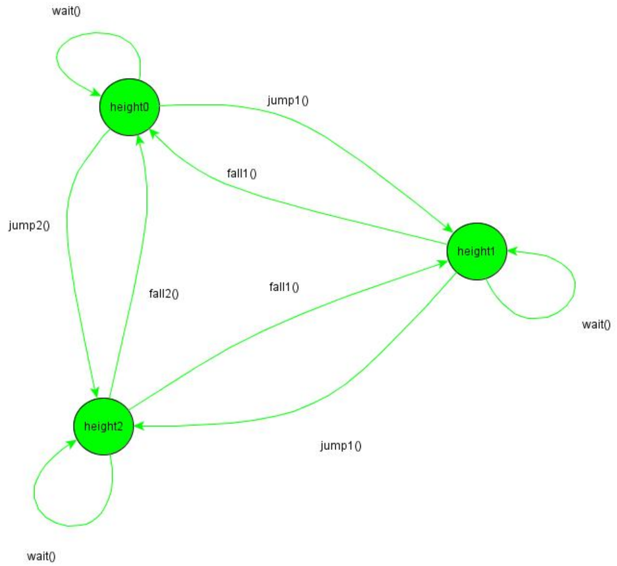
1. Bases on Breadth-First-Search
2. Starts at <u>every State</u>
3. Generate lists of all possible Input-Sequences, which have a given length, from any state.
4. Find a path to the start-State of an input-sequence
5. Add the path as an input-sequence to the calculated input-sequence
6. In the sample: N=3

All-N-Transitions Coverage Situation (I)



Input Lists

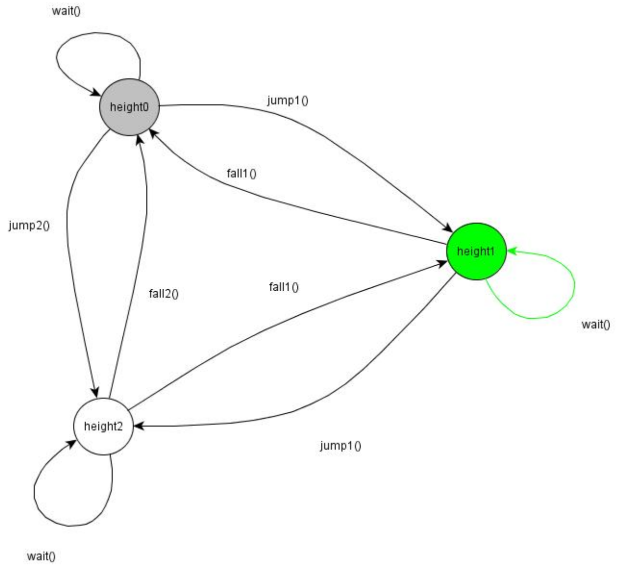
All-N-Transitions Coverage Situation (II)



Input Lists

1. wait()
2. jump1()
3. jump2()
4. wait()
5. jump1()
6. fall1()
7. wait()
8. fall1()
9. fall2()

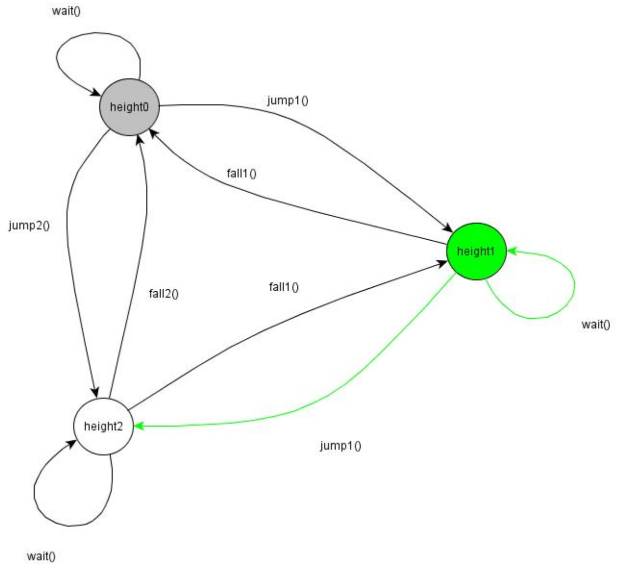
All-N-Transitions Coverage Situation (III)



Input Lists

1. wait()
2. ...

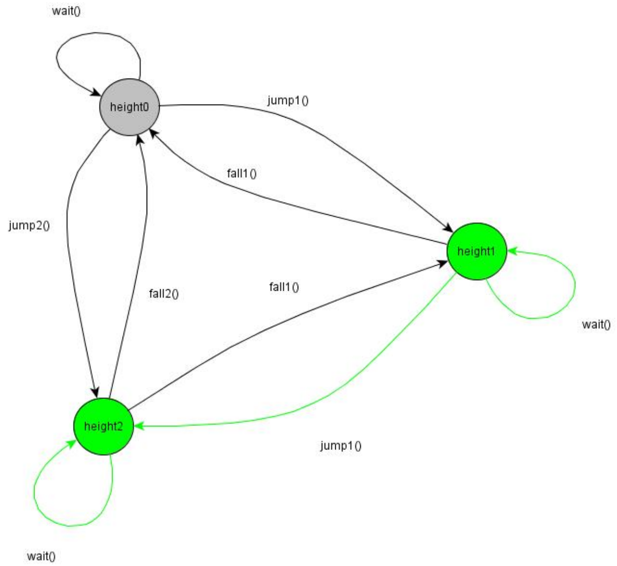
All-N-Transitions Coverage Situation (IV)



Input Lists

1. wait(), jump1()
2. ...

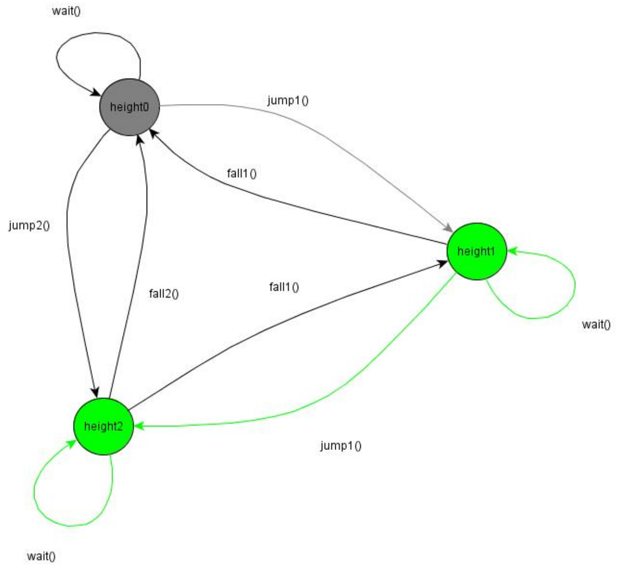
All-N-Transitions Coverage Situation (V)



Input Lists

1. wait(), jump1(), wait()
2. ...

All-N-Transitions Coverage Situation (VI)



Input Lists

1. jump1(), wait(), jump1(), wait()
2. ...

Further Issues

1. Event-Coverage : Suitable for a more complicated language like UML-Statecharts, where transitions can have multiple triggers.
2. Coverage-Criteria can be adapted to a language's features: Guards/Actions/Ports etc.
3. Test Implementation possibility 1 : While generating the Input-Sequence, save the visited states as well.
4. Test Implementation possibility 2 : Compare two implementations by processing every input in a stepwise manner.