Contents

1. Requirement-based testing

2. Dependence Analysis
   • Data/Control Dependence
   • Static/Dynamic Dependence Graph (SDG/DDG)
   • Static/Dynamic Interaction Pattern (SIP/DIP)

3. Test Suite Generation/Reduction using EFSM Dependencies
   • Equivalent Tests
   • SDG-based Test Suite Generation/Reduction
   • DDG-based Test Suite Reduction

4. TSGR Tools
   • STSG, STSR, DTSR, RTSR

5. Related Issues
Requirements Formalization

(Intended behavior)
Requirements

Informal
(e.g., English text)

A set of use cases

Formal
(e.g., EFSM or SDL)
An Example: ATM EFSM

1. **Start**
   - **Card** (pin, b)
   - Prompt for PIN; attempts = 0

2. **S1**
   - **PIN** (p)
   - (p != pin) and (attempts < 3)
   - Display error;
   - attempts = attempts + 1;
   - Prompt for PIN;

3. **S2**
   - **PIN** (p)
   - (p == pin)
   - Display menu

4. **T2**
   - Continue/Display menu

5. **T1**
   - **Withdrawal** (w)
   - b = b - w

6. **T6**
   - **Deposit** (d)
   - b = b + d

7. **T7**
   - **Balance**
   - print(b)

8. **T4**
   - **Deposit** (d)
   - b = b + d

9. **T9**
   - Exit/Eject card

10. **Exit**
    - **Deposit** (d)
    - b = b + d

11. **S3**
    - **Withdrawal** (w)
    - b = b - w

12. **T8**
    - Continue/Display menu

13. **T5**
    - **Balance**
    - print(b)

14. **T3**
    - Exit/Eject card

15. **Exit**
    - **Withdrawal** (w)
    - b = b - w

16. **S2**
    - **PIN** (p)
    - (p != pin) and (attempts == 3)
    - Display error;
    - Eject card;
    - Exit/Eject card

17. **T2**
    - **Withdrawal** (w)
    - b = b - w

18. **T1**
    - **Deposit** (d)
    - b = b + d

19. **T6**
    - **Balance**
    - print(b)
ATM Requirements

\[ R = \{ r \mid r \text{ is a requirement which is a transition} \} \]

or

\[ R = \{ r \mid r \text{ is a requirement which is a sequence of transitions} \}. \]

\[ R = \{ r_1, r_2, r_3, r_4, r_5 \} \]

for the ATM EFSM where:

\( r_1 \): User fails to enter a correct pin that matches with the PIN stored in the ATM card in less than four attempts. If pins do not match and less than four attempts were performed, the system displays an error message, increments the number of attempts and prompts for pin. At the fourth attempt, if user still fails to enter a correct pin, the system prints an error message and ejects the user’s ATM card.

\[ r_1 = T_2 T_2 T_2 T_3 \]

\( r_2 \): After entering a correct pin in less than four attempts, user selects withdrawal function. The system adjusts the balance and displays a menu with withdrawal, deposit, balance inquiry and exit functions.

\[ r_2 = x T_4 T_5 \]

\( r_3 \): After entering a correct pin in less than four attempts, user selects deposit function. The system adjusts the balance and displays a menu with withdrawal, deposit, balance inquiry and exit functions.

\[ r_3 = x T_4 T_6 \]

\( r_4 \): After entering a correct pin in less than four attempts, user selects balance function. The system displays the balance and then a menu with withdrawal, deposit, balance inquiry and exit functions.

\[ r_4 = x T_4 T_7 \]

\( r_5 \): After entering a correct pin in less than four attempts, user selects exit function and the system ejects the user’s ATM card.

\[ r_5 = x T_4 T_9 \]

\( x \): T2 may be inserted 0, 1, 2 or 3 times

\( T\# \) is the Transition Under Test (TUT)
1. Requirement-based Testing

Using the requirements given as an EFSM or SDL model in

- generation of a test suite or
- reduction of an existing test suite related to individual requirements

1. Different types of dependencies are identified between elements of the EFSM/SDL model of the requirements
2. Parts of the EFSM/SDL model that influence a requirement under test are identified using these dependencies
3. Equivalent tests are eliminated to generate or reduce the test suite.
2. Dependence Analysis

• Two tests are considered to be equivalent w.r.t. a Transition Under Test (TUT) if, during traversal of the EFSM/SDL model, these tests exhibit the same pattern of interactions w.r.t. the TUT.

• Two types of dependencies between transitions (requirements) in the EFSM/SDL model are identified as interactions among requirements:
  - Data Dependencies
  - Control Dependencies

• Patterns of interactions:
  - Static Interaction Pattern (SIP)
  - Dynamic Interaction Pattern (DIP)
Data Dependence

Data dependence captures the notion that one transition defines a value for a variable and some other or the same transition uses this value.

Let $T_i$ and $T_k$ be two transitions and $v$ be a variable in an EFSM/SDL model.

$(T_i, T_k, v)$ is a data dependence from $T_i$ to $T_k$ w.r.t. $v$ iff
- $v$ is defined in $T_i$ (as the last definition of $v$ in $T_i$)
- $v$ is used in $T_k$ (before $v$ is (possibly) defined in $T_k$)
- there is a def-clear path from $T_i$ to $T_k$ w.r.t. $v$.

A data dependence $(T_i, T_k, v)$ is a du-pair (def of $v$ in $T_i$, use of $v$ in $T_k$) w.r.t. $v$. 
Data Dependence

e.g.,

\[ T_1 \text{ defines } b \]
\[ T_5 \text{ uses } b \]
along \( T_1 \) \((T_2)\) \( T_4 \) \( T_5 \), \( b \) is not redefined.
\[ \therefore \text{ data dependence } (T_1, T_5, b) \text{ exists.} \]

\[ T_5 \text{ defines } b \]
\[ T_5 \text{ uses } b \]
along \( T_5 \) \( T_8 \) \( T_5 \), \( b \) is not redefined.
\[ \therefore \text{ data dependence } (T_5, T_5, b) \text{ exists.} \]
Control Dependence

- Control dependence captures the notion that one transition may influence the traversal of another transition.
- Control dependence between transitions is defined in terms of the concept of *post-dominance*. 
Post-dominance

Let $Y$ and $Z$ be two distinct states in an EFSM/SDL model and $T$ is an outgoing transition from $Y$:

- $Z$ post-dominates $Y$ iff $Z$ is on every path from $Y$ to the exit state.

- $Z$ post-dominates $T$ iff $Z$ is on every path from $Y$ to the exit state through transition $T$.

e.g.,
Take $S_1$ as $Y$ and $S_2$ as $Z$:
- $S_2$ does not post-dominate $S_1$
- $S_2$ post-dominates $T_4$
Control Dependence

Let $T_i$ and $T_k$ be two outgoing transitions from $Y$ and $Z$, respectively.

$(T_i, T_k)$ is a control dependence from $T_i$ to $T_k$ iff:
- $Z$ does not post-dominate $Y$
- $Z$ post-dominates $T_i$.

E.g.,
- $S2$ does not post-dominate $S1$
- $S2$ post-dominates $T4$.

$\therefore$ control dependence $(T4, T5)$ exists.
Data and control dependencies in the EFSM/SDL model are graphically represented in a Static Dependence Graph (SDG):
Patterns of Interactions

Patterns of interactions between functional elements of the system w.r.t. a TUT:

A **Static Interaction Pattern (SIP)** w.r.t. a TUT,
is a sub-graph of the SDG,
such that data and control dependence edges in SIP influence the TUT.

A **Dynamic Interaction Pattern (DIP)** of a test sequence w.r.t. a TUT,
is derived from the SDG,
via a **Dynamic Dependency Graph (DDG)**,
such that data and control dependence edges in DIP influence the TUT.
Static Interaction Patterns

e.g., suppose TUT = T5

A bounded set of SIPs can be constructed for a TUT in **SDG-based test suite generation** or a SIP can be derived from a test sequence w.r.t. a TUT in **SDG-based test suite reduction**.
Dynamic Interaction Patterns

e.g., suppose T1 T4 T5 T8 T5 T8 T9 is a test sequence for TUT = T5

A DIP is derived from a test sequence w.r.t. a TUT in **DDG-based test suite reduction**.
3. Test Suite Generation/Reduction

- There is no need to test the system with tests that exercise the same pattern of interactions w.r.t. a TUT. Such tests are equivalent tests.

- Two tests are equivalent w.r.t. the TUT if, during traversal of the EFSM/SDL model, these tests exhibit the same pattern of interactions w.r.t. the TUT.

- The goal is to reduce a test suite by identifying equivalent tests w.r.t. the TUT and removing all equivalent tests except one from the test suite.
Identification of Equivalent Tests

e.g.,

A test suite obtained w.r.t. the requirement represented by transition $T_5$, contains two tests among others:

- Test 1:
  - enter a valid pin,
  - perform a deposit transaction
  - and a withdraw transaction.

- Test 2:
  - enter an invalid pin then a valid pin,
  - perform a deposit transaction
  - and a withdraw transaction.

The number of times pin is entered before transition $T_5$ is executed, does not influence $T_5$.

Therefore, Test 1 and Test 2 are equivalent w.r.t. $T_5$. 
SDG-based Test Suite Generation

Given the set of requirements as an EFSM/SDL model and $R = \{ r \mid r \text{ is a requirement}\}$,

**Step 1:** Identify data and control dependencies between transitions of the EFSM/SDL model and generate the Static Dependence Graph (SDG) for the model.

**Step 2:** $\forall r \in R$, construct a set $S_r$ of $SIP_r$ using SDG where $SIP_r$ is a Static Interaction Pattern for $r$.

**Step 3:** $\forall r \in R$, construct $TS_r$ to cover each $SIP_r$ in $S_r$.

e.g., ATM model’s SDG and $R = \{r_1, r_2, r_3, r_4, r_5\}$

| $r$   | $|S_r|$ |
|-------|-------|
| $r_1$ | 1     |
| $r_2$ | 14    |
| $r_3$ | 14    |
| $r_4$ | 29    |
| $r_5$ | 1     |
SDG-based Test Suite Generation

e.g., for $r_2$, $S_{r_2}$ and $TS_{r_2}$ are:
SDG-based Test Suite Reduction

e.g., reduce a test suite for requirement \( r_2 \) where TUT is \( T_5 \) in the ATM model.

- \( r_2 \): After entering a correct pin in less than four attempts,
  user selects withdrawal function.
  The system adjusts the balance and displays a menu with withdrawal, deposit, balance inquiry and exit functions.
  
  \[ r_2 = x \ T_4 \ T_5 \]

- TUT is \( T_5 \).

- The test suite, \( TS_{T_5} \), generated by using the constrained selective path coverage test strategy with a constraint that each loop is traversed at most twice contains 18 tests.
### SDG-based Test Suite Reduction

\(TS_{T5}\):

<table>
<thead>
<tr>
<th>Test #</th>
<th>Test Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1 T4 T5 T8 T9</td>
</tr>
<tr>
<td>2</td>
<td>T1 T4 T5 T8 T5 T8 T9</td>
</tr>
<tr>
<td>3</td>
<td>T1 T2 T4 T5 T8 T9</td>
</tr>
<tr>
<td>4</td>
<td>T1 T2 T2 T4 T5 T8 T9</td>
</tr>
<tr>
<td>5</td>
<td>T1 T2 T4 T5 T8 T5 T8 T9</td>
</tr>
<tr>
<td>6</td>
<td>T1 T2 T2 T4 T5 T8 T5 T8 T9</td>
</tr>
<tr>
<td>7</td>
<td>T1 T4 T6 T8 T5 T8 T9</td>
</tr>
<tr>
<td>8</td>
<td>T1 T2 T4 T6 T8 T5 T8 T9</td>
</tr>
<tr>
<td>9</td>
<td>T1 T2 T2 T4 T6 T8 T5 T8 T9</td>
</tr>
<tr>
<td>10</td>
<td>T1 T4 T7 T8 T5 T8 T9</td>
</tr>
<tr>
<td>11</td>
<td>T1 T2 T4 T7 T8 T5 T8 T9</td>
</tr>
<tr>
<td>12</td>
<td>T1 T2 T2 T4 T7 T8 T5 T8 T9</td>
</tr>
<tr>
<td>13</td>
<td>T1 T4 T5 T8 T6 T8 T9</td>
</tr>
<tr>
<td>14</td>
<td>T1 T2 T4 T5 T8 T6 T8 T9</td>
</tr>
<tr>
<td>15</td>
<td>T1 T2 T2 T4 T5 T8 T6 T8 T9</td>
</tr>
<tr>
<td>16</td>
<td>T1 T4 T5 T8 T7 T8 T9</td>
</tr>
<tr>
<td>17</td>
<td>T1 T2 T4 T5 T8 T7 T8 T9</td>
</tr>
<tr>
<td>18</td>
<td>T1 T2 T2 T4 T5 T8 T7 T8 T9</td>
</tr>
</tbody>
</table>
SDG-based Test Suite Reduction

Given the set of requirements as an EFSM/SDL model, \( R = \{ r \mid r \text{ is a requirement} \} \) and a test suite \( TS \),

**Step 1**: Identify data and control dependencies between transitions of the EFSM/SDL model and generate the Static Dependence Graph (SDG) for the model.

**Step 2**: \( \forall r \in R \), identify TUT for \( r \) and form \( TS_{TUT} \).

**Step 3**: For each test suite \( TS_{TUT} \) and for each test sequence in \( TS_{TUT} \),

- derive a Static Interaction Pattern (SIP),
- then use it to identify equivalent tests in \( TS_{TUT} \).

1. Traverse the test sequence and identify data and control dependencies that are encountered during the traversal on SDG.
   Mark the identified dependencies in SDG.

For Test #2 in \( TS_{T5} \): T1 T4 T5 T8 T5 T8 T9
SDG-based Test Suite Reduction

2. Remove all unmarked dependencies from the SDG to obtain a dependence sub-graph.

3. Traverse the dependence sub-graph backward from the TUT and identify all dependencies that influence the TUT. Mark these identified dependencies in the dependence sub-graph.
SDG-based Test Suite Reduction

4. Remove unmarked dependencies from the dependence sub-graph to obtain a Static Interaction Pattern, where data and control dependencies represent interactions between transitions.

Two tests are equivalent w.r.t. a TUT if they result in the same Static Interaction Pattern.

Two tests are equivalent w.r.t. a TUT if they result in the same Static Interaction Pattern.
SDG-based Test Suite Reduction

$TS_{T5}$ contains initially 18 tests.

After applying the test suite reduction based on a Static Dependence Graph on $TS_{T5}$, three sets of equivalent tests are identified:

- Tests $\{1, 3, 4, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$ are all equivalent to test sequence $T1\ T4\ T5\ T8\ T9$ and result in SIP #1.
- Tests $\{2, 5, 6\}$ are all equivalent to test sequence $T1\ T4\ T5\ T8\ T5\ T8\ T9$ and result in SIP #2.
- Tests $\{7, 8, 9\}$ are all equivalent to test sequence $T1\ T4\ T6\ T8\ T5\ T8\ T9$ and result in SIP #3.
DDG-based Test Suite Reduction

Given the set of requirements as an EFSM/SDL model, \( R = \{ r \mid r \text{ is a requirement} \} \) and a test suite \( TS \),

**Step 1:** Identify data and control dependencies between transitions of the EFSM/SDL model and generate the Static Dependence Graph (SDG) for the model.

**Step 2:** \( \forall r \in R \), identify TUT for \( r \) and form \( TS_{TUT} \).

**Step 3:** For each test suite \( TS_{TUT} \) and for each test sequence in \( TS_{TUT} \),

derive a Dynamic Interaction Pattern (DIP), then use it to identify equivalent tests in \( TS_{TUT} \).

1. During traversal of the test sequence,

identify data and control dependencies on SDG and create a Dynamic Dependence Graph (DDG) for the test sequence.

Each transition is represented as a separate node in DDG.

For Test #2 in \( TS_{TS} \): T1 T4 T5 T8 T5 T8 T9

Dynamic Dependence Graph for Test #2 in \( TS_{TS} \)
DDG-based Test Suite Reduction

2. All dependencies that influence the TUT are identified by traversing the DDG backward from the TUT and marking all the traversed dependencies.

Dynamic Dependence Graph for Test #2 in $TS_{T5}$

3. Unmarked dependencies are removed to obtain a Dynamic Interaction Pattern, where data and control dependencies represent interactions between transitions.

Dynamic Interaction Pattern for Test #2 in $TS_{T5}$

Two tests are equivalent w.r.t. a TUT if they result in the same Dynamic Interaction Pattern.
DDG-based Test Suite Reduction

$T_{S_{T5}}$ contains initially 18 tests.

After applying the test suite reduction based on a Dynamic Dependence Graph on $T_{S_{T5}}$, three sets of equivalent tests are identified:

- Tests $\{1, 3, 4, 10, 11, 12, 13, 14, 15, 16, 17, 18\}$ are all equivalent to test sequence $T1 \ T4 \ T5 \ T8 \ T9$ and result in DIP #1.
- Tests $\{2, 5, 6\}$ are all equivalent to test sequence $T1 \ T4 \ T5 \ T8 \ T5 \ T8 \ T9$ and result in DIP #2.
- Tests $\{7, 8, 9\}$ are all equivalent to test sequence $T1 \ T4 \ T6 \ T8 \ T5 \ T8 \ T9$ and result in DIP #3.
4. TSGR Tools

*TSGR* is a set of software tools developed in Solaris, applying the (SDG/DDG)-based test suite generation/reduction to generate a test suite or to reduce an existing test suite (TS) w.r.t. a set of requirements (R) of a system modeled as an EFSM.

Some information can be found at: http://www.site.uottawa.ca/~ural/
STSG Tool

**STSG (Static Test Suite Generation)** tool takes as input,

- an EFSM modeling the system under test and
- a set of requirements (R) in form of a set of TUTs.

Then, for each TUT,

STSG applies the SDG-based test suite generation
constructing the Static Interaction Patterns w.r.t. the TUT and
arbitrarily constructs one test for each SIP.

Finally, STSG produces as output,

- the SIP-based test suite (RTS) w.r.t. R.
STSR and DTSR Tools

STSR (Static Test Suite Reduction) tool takes as input,
- an EFSM modeling the system under test,
- an existing test suite (TS) and
- a set of requirements (R) in form of a set of TUTs.

Then, for each TUT,
STSR identifies the tests in TS, which are applicable to the TUT,
applies the SDG-based test suite reduction
deriving the SIP of each test sequence w.r.t. the TUT and
arbitrarily selects only one test from each set of equivalent tests.
Finally, STSR produces as output,
- the SIP-based reduced test suite (RTS) w.r.t. R.

DTSR (Dynamic Test Suite Reduction) tool takes as input,
- an EFSM modeling the system under test,
- an existing test suite (TS) and
- a set of requirements (R) in form of a set of TUTs.

Then, for each TUT,
DTSR identifies the tests in TS, which are applicable to the TUT,
applies the DDG-based test suite reduction
deriving the DIP of each test sequence w.r.t. the TUT and
arbitrarily selects only one test from each set of equivalent tests.
Finally, DTSR produces as output,
- the DIP-based reduced test suite (RTS) w.r.t. R.
RTSR (Regression Test Suite Reduction) tool takes as input,

- an EFSM modeling the system under test,
  along with the changed (added, deleted, modified) transitions specifying the changes in requirements,
- an existing test suite (TS) and
- a set of requirements (R) in form of a set of TUTs.

Then, for each TUT,
RTSR identifies the tests in TS, which are applicable to the TUT,
applies the SDG-based regression test suite reduction (not in course notes) deriving the SIP of each test sequence w.r.t. the TUT and arbitrarily selects only one test from each set of equivalent tests.

Finally, RTSR produces as output,
- the SIP-based reduced regression test suite (RTS) w.r.t. R.
5. Related Issues

- Test reduction using Static Interaction Patterns is appropriate in the initial stages of testing, when a relatively small number of tests are supposed to be used. However, test reduction using Static Interaction Patterns ignores repetitions of the same dependencies (interactions) between transitions.

- The size of the reduced test suite based in Dynamic Interaction Patterns may not be bounded. It depends on a selected testing strategy and the length of test sequences.

- Since any test suite reduction may lead to reduction in test suite fault detection capability. Experiment should be performed to investigate the degree of reduction in fault detection capability of reduced test suite when the presented approach of test reduction is used.

- Test equivalency should be more efficiently determined at low cost, specially when test sequences are long.

- Regarding STSG tool, semantics of the statements in each transition should be considered so that only feasible tests are generated in RTS.