ISSN 0361-7688, Programming and Computer Software, 2016, Vol. .., No. .., pp. ...
Original Russian Text .....published in Proceedings of the Institute for System Programming of the Russian Academy of Sciences 2015

A Stream X-Machine tool for modelling and generating test cases for chronic diseases based on state-counting approach

Authors
Organisation.
e-mail: {email}@email
Received .....

Abstract—In the biomedical domain, diagrammatical models have been extensively used to describe and understand the behaviour of biological organisms (biological agents) for decades. Although these models are simple and comprehensive, they can only offer a static picture of the corresponding biological systems with limited scalability. As a result, there is an increasing demand to integrate formalism into more dynamic forms that can be more scalable and can capture complex time-dependent processes. Stream X-Machine (SXM) is such a powerful formal method with a memory (data) structure and function-labelled transitions. One of the main strengths of the SXM is its associated testing strategy which ensures that, under well-defined conditions, all functional inconsistencies between the system under test and the model are revealed. In this paper, we adopt the concept of SXM to develop a tool known as T-SXM, which has the capabilities of modelling real world problems and generating test cases automatically based on the state-counting approach. The Type II diabetes case study has been used to demonstrate the abilities of the proposed tool.

DOI:

1. INTRODUCTION

Human body can be recognised as a complex system where constituent sub systems such as immune system, digesting system, cardiovascular system, etc. work together to keep the body active and healthy. As any other systems, human body also faces issues (errors or undesired state) namely diseases. Due to the complex inter-relationship of the constituent systems of human body, the diseases also demonstrate the same complex interconnections with the constituent systems of the body and as well as with the diseases themselves [1]. New omics sciences have identified that even though the diseases can be generalised, they act differently from individual to individual ([2] [3]).

Ferguson et al. [2] recognised that evidence-based personalized or precision medicine through dietary intervention could add significant value on the healthcare sector which has a considerable potential in curing chronic diseases and improving healthy living. Evidence exists that dietary interventions have significant impact in genetical disorders, such as inborn errors of metabolism (e.g. phenylketonuria mutations in human leukocyte antigen complex and other genes causes celiac disease or gluten sensitivity or variants in the lactate gene affecting lactate persistence) [2] [3] [4].

A chronical medical condition or disease is classified as “physical or mental health condition that lasts more than one year and causes functional restrictions or requires ongoing monitoring or treatment” [5] [6]. Managing patients with chronical diseases has changed from single dimension (medicinal) to multidimensional approach. The state-of-the-art approaches in managing people with chronical diseases have now become more lifestyle modification, regular physical activities and especially nutritional and dietary cantered approach [7].

At present, the medical professionals are focusing on precise medicine and alternative practices, such as nutritional interventions and behavioural changes in treating chronical diseases [8]. Currently there are very limited recourses/tools available to support this approach. Therefore, this has limited the ability of the healthcare professionals using this approach to provide their service to their patients [8] [9] [10].

Biological systems (e.g., human body) consist of closely connected components (organs) that change their actions and behaviours over the time and their interactions with exposure to external factors (e.g., nutrients, viruses, bacteria). In addressing such challenges, literature reveals that, there are few formal approaches has been occupied namely, Boolean Networks (BN) and its extensions (i.e. Qualitative Networks (QN), Gene Regulatory Networks (GRN) [11] [12] [13], Petri Nets (PR), Cellular Automata (CA), Population P systems (PPS), etc.

Approximately one third of the world adult population is suffering from Multiple Chronic Conditions (MCCs) [14]. Therefore, researchers have also adopted Communicating X-Machine (CXM), a formal specification method, to model biological systems where it has given the ability of representing the inter-relationships and the communication aspects of such systems [15]. To the best of our knowledge, this is the first attempt that uses a formalism technique to model a chronic disease.

Beside CXM, SXM is also a formal specification technique and is the most well-known variance of the X-Machine introduced by Eilenberg [16]. SXM is supposed to resolve a problem that exists in the original X-Machine which is the lack of the ability to process sequences of inputs and outputs. According to Dranidis et al. [17], the powerful modelling capabilities of SXMs have been applied in a number of research projects such as the EURACE, SUMO, and Epitheleome for the simulation of cellular and social systems. Additionally, one of the great benefits of using an SXM to specify a system is its associated testing method which was initially developed for deterministic SXM [18] [19] and was further extended to non-deterministic SXM [20] and communicating SXM [21]. Under certain design-for-test conditions, this method can produce a test suite that can be used to verify the correctness of the SUT provided that the processing functions of the SXM specification have been correctly implemented [22].

Although the effectiveness of the method has been validated by a wide range of industrial case studies [23], the application of the SXM based testing method is often encumbered due to the strictness of the design for test conditions. There have been several improvements to the SXM testing method with the aim to relax the design-for-test conditions [19] [21] [22] [24]. Still, there exists a limited number of tools that demonstrate the practical benefits of the SXMs.

In this paper, we propose the T-SXM, a new and, to the best of our knowledge, the only existing tool that uses Java as the language to describe the behaviours of a system (a.k.a. formal specification) and supports automated test generation based on the state-counting approach [24]. This tool is an improved version of the tool we proposed in [25]. We use the Type II diabetes as a case study to evaluate the performance of the T-SXM tool in terms of modelling a chronic disease and generating test cases.

The rest of the paper is organised as follows. Section 2 provides an overview of the techniques that have been used to model chronic diseases. Section 3 defines the formalism of SXM. Section 4 discusses the SXM testing method and the improvements of the method to relax the design-for-test conditions. Section 5 introduces the T-SXM tool with related algorithms. Section 6 presents the Type II diabetes case study. Section 7 discusses the evaluation of T-SXM in terms of modelling and generating test cases for the case study. Section 8 outlines and compares T-SXM with the existing related tools for SXMs. Finally, section 9 concludes the paper and presents the future research directions.

2. DISEASE MODELS IN THE LITERATURE

There have been many attempts and research carried out in the scope of disease prediction based on biomedical data. However, most of these approaches have not been able to provide 100% accurate analysis and, in a safety-critical domain like human medicine this poses a serious risk as consequences of an error could cause a life-threatening result [26] [27].

BN considers the coarse approximation in which the individual models have two states namely active and inactive without considering intermediary states. However, in reality, biological systems consist of multiple states and in order to facilitate that, the QN approach has been proposed as an extension for the BN where each variable can have a small number of discreate variable values, and the dependencies of those values are expressed with algebraical functions instead of Boolean functions. This poses a challenge in modelling diseases as they consist of many different states which are interconnected with complex relationships [11] [12] [13].

PR provides a very comprehensive modelling and analysing facility for distributed and concurrent systems such as biological systems. PR has the ability in modelling non-deterministic systems. Compared to BN, PN has a good balance between modelling power and the analysability [28]. Hence, the ability of providing the concurrency the complexity of the model increases significantly. While subclasses of PR increase the decision-making power, they limit the ability of modelling larger system due to complexity. Nevertheless, PR suffers from the inability to test in unbounded places [29].

CA [30] provides a means of modelling interactive components on a system such as biological systems. CA consists of an array of cells or agents which has a predefined communication with the neighbouring agent. Each agent or the cell maintains a state and a logical operation for the next state on the agent and as well as for the neighbouring agent. Each agent has been formally defined as a finite state machine (FSM). CA is a powerful way of defining agent-based system due to the simplicity of following rules and ease of verification [30]. However, CA presents challenges in modelling non-trivial systems due to lack of data representation. When the additional complexity adds to an agent, the neighbouring agents' number of states and input symbols increase significantly, which leads to state and input symbol explosion. Furthermore, the agents are aligned with a static grid for communication, therefore mobility of agents are limited and the ability to communicate with different agents is restricted. This presents an enormous challenge in modelling diseases as the communications and the iterations cannot be predefined due to the uniqueness of the individual circumstances.

PPS defines a system as an arbitrary graph. Each node of the graph contains membrane, that assigned to multiset of objects combine with set of rules that uses to modify the object and communicating is defined as edge of the graph. In the perspective of the biological systems, this can be interpreted as abstract entities of bio-agents (medical and/or diseases) which aggregate to form a complex system (human). PPS provides a robust mechanism to introduce new nodes, remove nodes, and change the behaviour of the defined nodes. On the other hand, PPS lacks the ability to represent internal states and individual behaviour of the nodes [15]. This indicates a significant challenge in defining the individual diseases (e.g., the complexity of the disease models encompasses a significant number of input symbols and states which need to be monitored and manipulated closely).

Due to the above-mentioned reasons, many mathematical models present significant challenges in modelling diseases due to the highly complex relationship(s) among the disease models. As the behaviours of the model are highly complex, most of the mathematical models have been challenged and become unmanageable [26]. Therefore, Wang et al. [28] recommended the use of formalism approaches for modelling the disease models. Specifically, SXM models will be more appropriate due to their power of modelling real world systems where the data structure and controllers need to be managed separately.

3. STREAM X-MACHINE

An X-Machine [16] is an enhanced version of an FSM with a basic data set, X, and a set of processing functions, Φ, which operate on X. An X-Machine can potentially model very general systems as the data set X can contain information about the system internal memory as well as different output behaviours.

A number of classes of X-Machines have been identified and studied [31]. Among these classes, the stream X-Machine (SXM) has received the most attention. The SXM is supposed to resolve a problem that exists in the original X-Machine which is the lack of the ability to process sequences of inputs and outputs. In this section, the Stream X-Machine and its related basic concepts are defined.

Definition 1. A Stream X-Machine is a tuple:

$$Z = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$$

where:

- Σ is a finite set of input symbols,
- Γ is a finite set of output symbols,
- Q is a finite set of states,
- M is a (possibly) infinite set called memory,
- Φ is a finite set of partial functions φ (processing functions) that map memory-input pairs to output-memory pairs, $\varphi: M \times \Sigma \rightarrow \Gamma \times M$,
- F is the next-state partial function, $F: Q \times \Phi \rightarrow Q$
- $q_0 \in Q$ and $m_0 \in M$ are the initial state and initial memory respectively.

Intuitively, an SXM can be thought as a finite automaton with the arcs labelled by functions from the type Φ. The automaton $A_Z = (\Phi, Q, F, I, T)$ is called *the associated finite automaton (FA)* of Z and is usually described by a state-transition diagram.

Definition 2. An SXM Z is called *deterministic* if the following conditions are met.

- The associated FA is deterministic:
  - Z has only one initial state: $I = \{q_0\}$;
  - The next state function of Z maps each pair (state, processing function) onto at most one state: $F: Q \times \Phi \rightarrow Q$;
- Any two distinct processing functions that label arcs emerging from the same state have disjoint domains: $\forall \varphi_1, \varphi_2 \in \Phi, ((\exists q \in Q \text{ with } (q, \varphi_1), (q, \varphi_2) \in dom(F)) \Rightarrow (\varphi_1 = \varphi_2 \text{ or } dom(\varphi_1) \cap dom(\varphi_2) = \emptyset))$

An SXM is deterministic when there is exactly one transition for any triplet $q \in Q$, $m \in M$, $\sigma \in \Phi$.

Definition 3. For $q \in Q$, the *language accepted* by Z in q, denoted by $L_Z(q)$, is defined by:

$$L_Z(q) = \{s \in \Sigma^* \mid (q, s) \in dom(F^*)\}.$$

The language accepted by Z in q_0 is simply called the *language accepted by* Z and is denoted by LA_Z.

Definition 4. A state $q \in Q$ is called *accessible* if $\exists s \in \Sigma^*$ with $F^*(q_0, s) = q$. Z is called *accessible* if $\forall q \in Q$, q is accessible.

Definition 5. For $U \subseteq \Sigma^*$, two states q_1 and q_2 are called *U-equivalent if* $L_Z(q_1) \cap U = L_Z(q_2) \cap U$. Otherwise, q_1 and q_2 are called *U-distinguishable*. If $U = \Sigma^*$, then q_1 and q_2 are simply called *equivalent* or *distinguishable*. Z is called *reduced* if $\forall q_1, q_2 \in Q$, q_1 and q_2 are distinguishable.

Definition 6. A deterministic stream X-Machine, Z, is called *minimal* if and only if Z is *accessible* and *reduced* [16].

In what follows, we only consider ***minimal deterministic SXM (DSXM)*** specifications.

4. STREAM X-MACHINE FOR TESTING

The testing method was developed for SXM specifications that meet two design for test conditions: *output-distinguishability* and *input-*

completeness (controllability) [18]. The first requires that every processing function can be distinguished by examining the output produced when an input is applied to any given memory value. Controllability basically means that every path in the associated automaton can actually be driven by suitable input sequences. Whilst the first condition is quite natural and can be satisfied by a suitable enrichment of the observed output, controllability is seldom met by non-trivial specifications. Therefore, in [24], the original testing method is generalised by replacing the input-completeness condition by a laxer condition, called *input-uniformity*.

4.1. *Reaching and distinguishing states in DSXM*

A. *Realisable sequences*

As the labels used in the state-transition diagram of a DSXM are actual functions, there may be states that are reachable in the diagram but cannot be reached in practice by any input sequence [24]. Similarly, there may be pairs of distinguishable states in the associated FA for which the sequences of processing functions that distinguish them can never be applied [24].

To determine which states can practically be reached or distinguished, it is essential to identify which sequences of processing functions in the associated FA can be driven by the input sequences from each state q and memory value m.

Definition 7. The set $R_\Phi(m) \subseteq \Phi^*$ is defined to consist all sequences of processing functions $p = \varphi_1 \cdot \cdot \cdot \varphi_n \in \Phi^*$, $n \geq 0$, for which there exists $s = \sigma_1 \cdot \cdot \cdot \sigma_n \in \Sigma^*$ such that $(m, s) \in dom\|p\|$. Then, $LR_Z(q, m) = LA_Z(q) \cap R_\Phi(m)$ and $LR_Z = LA_Z \cap R_\Phi(m_0)$.

B. *r-reachable states*

It is possible to reach some states of a DSXM with sequences in LR_Z using appropriate input sequences. Such states are said to be *r-reachable*.

Definition 8. State q of Z is said to be r-reachable if there exists $p \in LR_Z$ such that $F^*(q_0, p) = q$.

Any states that are not r-reachable can be removed from the machine without affecting the function computed by the machine. Since $\varepsilon \in LR_Z$, the initial state is always r-reachable.

An *r-state cover* of Z is a minimal set of realisable sequences $S_r \subseteq LR_Z$, $\varepsilon \in S_r$, that reaches every r-reachable state in Z.

Definition 9. A set $S_r \subseteq LR_Z$ is called an r-state cover of Z if:

- $\varepsilon \in S_r$
- For every r-reachable state q of Z, there exists $p \in S_r$ such that $F^*(q_0, m) = q$.

For every two distinct sequences $p_1, p_2 \in S_r$, $F^*(q_0, p_1) \neq F^*(q_0, p_2)$.

C. *Separable states*

States in the specifications can be distinguished by applying a finite set of realisable sequences of processing functions to their current memory values [24]. More formally, the set $MAtt(q)$ of attainable memory values in state q is defined to consist all memory values computed along all sequences in LR_Z that reach q.

$m \in MAtt(q)$ if there exists $p \in LR_Z$, $s \in \Sigma^*$, $g \in \Gamma^*$ such that $F^*(q_0, p) = q$ and $\|p\|(m_0, s) = (g, m)$.

States q_1 and q_2 are said to be *r-distinguishable* if there exists a *finite set* of sequences Y such that for every $m_1 \in MAtt(q_1)$ and every $m_2 \in MAtt(q_2)$, $LR_Z(q_1, m_1) \cap Y \neq LR_Z(q_2, m_2) \cap Y$.

Based on [22], it can be said that r-distinguishability is sufficient when the DSXM model of the implementation is known to be controllable. On the other hand, a stronger condition, called separability, is required which ensures that states are r-distinguished by sequences with overlapping domains [32].

Definition 10. States q_1 and q_2 are said to be *separable* if there exists a finite set of sequences

Y such that for every $m_1 \in MAtt(q_1)$ and every $m_2 \in MAtt(q_2)$, there exists $p_1 \in LR_Z(q_1, m_1) \cap Y$ and $p_2 \in LR_Z(q_2, m_2) \cap Y$ such that $p_1 \neq p_2$ and $dom\|p_1\| \cap dom\|p_2\| \neq \emptyset$. Y is said to separate between q_1 and q_2.

Definition 11. A *separating set* $W_s \subseteq \Phi^*$ of Z is a set of sequences of processing functions that separates between every pair of separable states of Z.

4.2. *Design for test conditions*

Definition 12. Φ is said to be *output-distinguishable* if for all $\varphi_1, \varphi_2 \in \Phi$, whenever there exists $m, m_1, m_2 \in M$, $\sigma \in \Sigma$, $\gamma \in \Gamma$ such that $\varphi_1(m, \sigma) = (\gamma, m_1)$ and $\varphi_2(m, \sigma) = (\gamma, m_2)$, then $\varphi_1 = \varphi_2$.

With the output-distinguishability condition, testers can determine the sequence of processing functions applied in the implementation under test (IUT) by examining the output sequence produced when an input sequence is applied.

Whilst the output-distinguishability condition is quite natural and can be satisfied by a suitable enhancement of the observed output, the input-completeness condition is seldom met by non-trivial SXM specifications [22]. This condition is normally enforced by designing extra input symbols to trigger the functions. Nevertheless, these additional inputs must be removed after testing has been completed and thus, it can potentially lead to extra errors in the specification. In [24], the input-completeness was slightly relaxed, being replaced by the input-uniformity condition which also suffers from the same limitations [22]. Therefore, in this paper, ***the input-completeness or input-uniformity condition is no longer required in the specification***.

4.3. *Reduced test function*

A test generation process includes two phases: initially, test cases are derived in the form of sequences of processing functions and later on, these sequences of processing functions are transformed into input sequences which are also referred to as the actual test data [22].

A test function is a function of the general form $t: \Phi^* \rightarrow \Sigma^*$ that satisfies the conditions which will be described later. Previous SXM based testing methods ([18] [24] [32]) relied on the test function that appends to the longest realisable prefixes of the processing function sequence, an extra input that causes the invocation of a non-existing transition in the associate FA. In [22], a reduced test function was introduced to reduce the length of the test sequences.

In what follows, we first present the definition of the original test function and then, we present the definition of the reduced test function.

Definition 13. A test function of an SXM Z is a function $t: \Phi^* \rightarrow \Sigma^*$ that satisfies the following conditions:

- $t(\varepsilon) = \varepsilon$. (1)
- Let $p = \varphi_1 \cdots \varphi_k \in \Phi^*$, $k \geq 1$.
    - If $\varphi_1 \cdots \varphi_{k-1} \in LA_Z$ and $\varphi_1 \cdots \varphi_k \in R_\Phi(m_0)$, then $t(p) = \sigma_1 \cdots \sigma_k$ for some $\sigma_1, \ldots, \sigma_k$ such that $(m_0, \sigma_1 \cdots \sigma_k) \in dom\|p\|$; (2)
    - Otherwise, $t(p) = t(\varphi_1 \cdots \varphi_{k-1})$. (3)

The first rule says that an empty path is transformed into an empty sequence. The second rule states that if the longest proper prefix $\varphi_1 \cdots \varphi_{k-1}$ is a path in A_Z then $t(p)$ is an input sequence, if exists, that drives p. Eventually, when $\varphi_1 \cdots \varphi_{k-1}$ is not a path in A_Z or no such input sequence exist, the construction of $t(p)$ is recursively reduced to the construction of $t(\varphi_1 \cdots \varphi_{k-1})$.

Definition 14. A *reduced test function* of an SXM Z is a function $\tau: \Phi^* \rightarrow \Sigma^*$ that satisfies the following conditions:

- $\tau(\varepsilon) = \varepsilon$. (1)
- Let $p = \varphi_1 \cdots \varphi_k \in \Phi^*$, $k \geq 1$.
    - If $\varphi_1 \cdots \varphi_{k-1} \in LR_Z$, then $\tau(p) = \sigma_1 \cdots \sigma_k$ for some $\sigma_1, \ldots, \sigma_k$ such that $(m_0, \sigma_1 \cdots \sigma_k) \in dom\|p\|$; (2)
    - Otherwise, $\tau(p) = \tau(\varphi_1 \cdots \varphi_{k-1})$. (3)

As above, if $p = \varphi_1 \cdots \varphi_k$ is a path in A_Z then $\tau(p)$ is a sequence of inputs that drives p, if there exists. Otherwise, if $p = \varphi_1 \cdots \varphi_k$ is not a path in A_Z or there exists no such an input sequence, then τ returns an input sequence that drives the longest prefix of p that is a path in AZ (without extending this prefix with an input to check the first non-existing arc, like in the case of t. In this paper, we will be ***using the reduced test function to generate input sequences for a processing function sequence***.

4.4. *Deriving integration test sequences through state-counting*

The state-counting approach was described in detail in [22] and [24] with intuitive examples. In this paper, we summarise the key values and sets used in equation (1) which calculates the test suite U:

$$U = \bigcup_{q \in Q_r} \{p_q\} pref(V(q)) W_s \qquad (1)$$

In order to construct the test suite U, we first need to select two sets of sequences of processing functions, S_r and W_s, and of a relation d_s on the states of Z as follows:

- S_r is a non-empty set of realisable sequences such that *no state in Z is reached by more than one sequence in* S_r.
- W_s is a finite set that separates between separable states of Z. W_s is required to be non-empty.
- d_s: $Q \longleftrightarrow Q$ is a relation on the states of Z that satisfies the following conditions: for every two states $q_1, q_2 \in Q$, if $(q_1, q_2) \in d_s$ then q_1 and q_2 are separated by W_s. The relation (d_s) identifies pairs of states that are known to be separated by W_s.
- The maximal sets $Q_1,\ldots,Q_j$ of states of Z that are known to be pairwise separated by W_s.

The definition of the set $V(q)$ can be found in [24].

For any choice of S_r, W_s, and d_s and for every $q \in Q_r$, the set $V(q)$ can be computed [24]. Once the $V(q)$ sets have been constructed, a test suite can be generated by taking all sequences in $\{p_q\}pref(V(q))$, concatenating them with W_s and applying the reduced test function τ: $\Phi^* \rightarrow \Sigma^*$ to every resulting sequence of processing functions.

For any choice of S_r, W_s, and d_s, the set U is finite and computable [24].

The test suites generated using the state-counting approach may be significantly larger than those generated by the W-method. However, the size of the test suite and the complexity of the test generation algorithm can be significantly reduced by designing the SXM specification in which all states are pairwise r-distinguishable [24].

5. THE T-SXM TOOL

In this section, we present the T-SXM tool, which implements the strength of the SXM-based testing strategy.

T-SXM is a tool, developed in Java, that allows the specification of SXM models, and most importantly, the automated test generation.

5.1. *The package and class diagram*

Fig. 1 shows the class diagram which is the design for the T-SXM tool, where:

- **SXMBase**: the base data model of an SXM,
- **SXM**: the extension of SXMBase with testing functions,
- **State**: the state of the SXM,
- **Memory**: the memory of the SXM,
- **Function**: the processing function of the SXM,
- **Transition**: the transition of the SXM,
- **Variable**: the input and output alphabet of the SXM,
- **DataType**: the data type of the SXM,
- **InputType**: the input type of the SXM,
- **OutputType**: the output type of the SXM.

- **InputMemoryPair**: the input – memory tuple,
- **OutputMemoryPair**: the wrapper class for the output – memory tuple that a processing function should return after completing the computation.

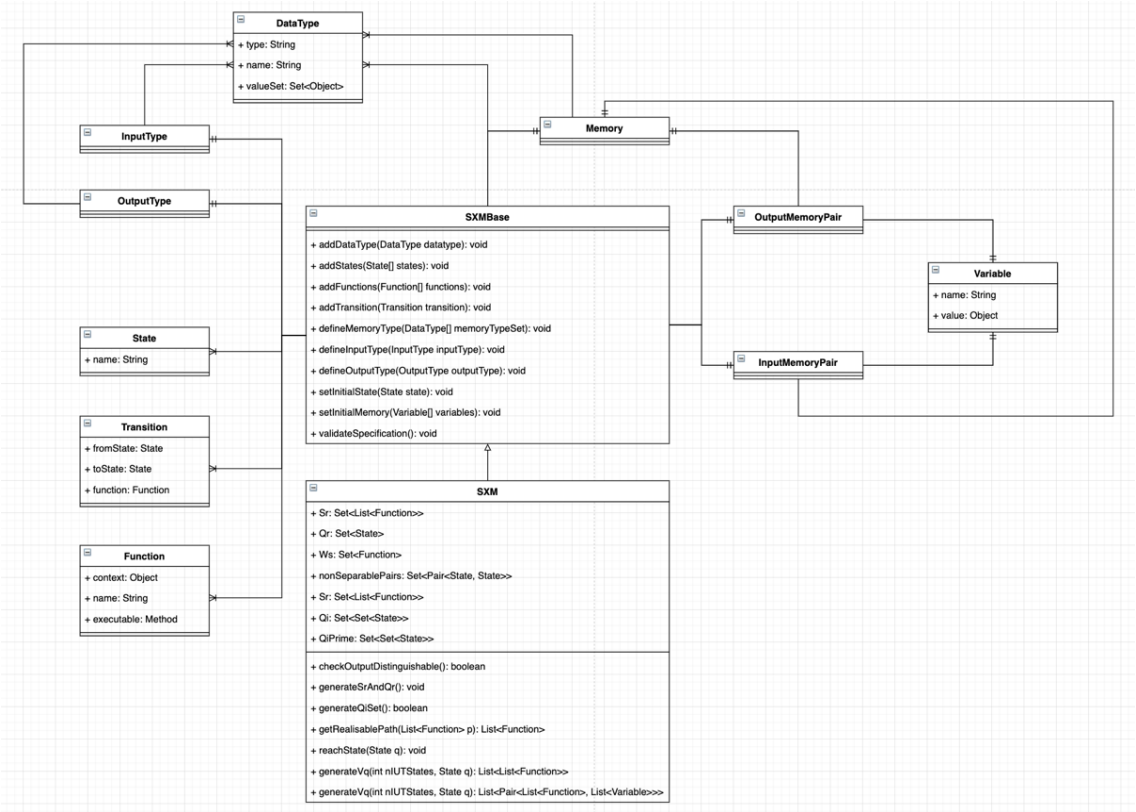

Fig. 1. T-SXM class diagram

5.2. *The related algorithms*

This section outlines the algorithms that are used to validate an SXM specification and derive the key values and sets of equation (1) discussed in section II.C.4.

A. *Algorithm for checking output-distinguishability condition*

As output-distinguishability is one of design for test conditions, we developed a dedicated algorithm to ensure the specification meets this condition.

```
Input: None
Output: boolean
-------------------------------------------------------------------------

possibleInputs ← all combinations of possible inputs
possibleMemories ← all combinations of possible memory values
functionSet ← set of all processing functions

pairsOfNonDistinguishableFunctions = {}

While possibleMemories != {}:
        While possibleInputs != {}:
```

```
                        pm ← get memory element
                        pi ← get input element

                        outputsForThisInput = {}

                        While functionSet != {}:
                                f ← get function
                                If F(pi, pm) ∈ domain(f), then:
                                        outputsForThisInput
= outputsForThisInput ∪ Pair(f, output)

                        For every Pair(f1, output1), (f2, output2) ∈
outputsForThisInput:
                                If there exists any f1 != f2 and
output1 = output2, then:

                pairsOfNonDistinguishableFunctions =

                pairsOfNonDistinguishableFunctions ∪ Pair(f1, f2)

If PairsOfNonDistinguishableFunctions = {}, return true.
Otherwise, return false.
```

B. *Algorithm for generating S_r and Q_r*

S_r and Q_r are two of the key sets required in equation (1) discussed in section II.C.4. This algorithm is designed to identify all the r-reachable states in the SXM specification and construct the S_r and Q_r sets accordingly. Also, all the states that are not r-reachable along with the arcs (transitions) emerging from or arriving to them will be removed.

```
Input: None
Output: None (this function will directly update the Sr and Qr sets in the
machine)
--------------------------------------------------------------------------

Sr = {}
Qr = {}
possibleInputs ← all combinations of possible inputs
possibleMemories ← all combinations of possible memory values
stateSet ← set of all states

While stateSet != {}:
        q ← get state
        For all pi ∈ possibleInputs, pm ∈ possibleMemories:
                If there exists a sequence of functions, p, that
actually reach q, then:
                        Qr = Qr ∪ q
                        Sr = Sr ∪ p

Remove all states that are not r-reachable
Remove all arcs (transitions) emerging from or arriving to those states.
```

C. *Algorithm for generating Q_i set*

This algorithm is the solution for constructing the Q_i sets.

```
Input: Ws, nonSeparablePairsOfStates
Output: None (this function will directly update the Qi set in the
machine)
--------------------------------------------------------------------------

Qi = {}

If nonSeparablePairsOfStates = {}:
        Qi = Qi ∪ Qr
Else:
        For each pair(q1, q2) ∈  nonSeparablePairsOfStates:
                setOne = {q1}
                setTwo = {q2}

                For each q ∈ Qr:
                        If q and q1 are pairwise
separated:
                                setOne = setOne ∪
q
                        If q and q2 are pairwise
separated:
                                setTwo = setTwo ∪
q

                If setOne does not exist in Qr:
                        Qi = Qi ∪ setOne

                If setTwo does not exist in Qr:
                        Qi = Qi ∪ setTwo
```

D. *Algorithm for computing V(q)*

Based on the key values and sets which are computed by algorithms *b* and *c*, we can derive the *V(q)* using this algorithm.

```
Input: k ← estimated number of states in the IUT, state q
Output: Vq
--------------------------------------------------------------------------

Vq = {}
stateCountQis = {n0 = 0, n1 = 0,..., nj = 0} ← j = length of Qi set

Y = {Triplet(<>, q, stateCountQis)} ← list of triplets each of which
contains 3 elements: function sequence, state, and stateCountQis

While Y != {}:
        y = Y.remove(0)
        yFunctionSequence = y.getValue0()
        yState = y.getValue1()
        yStateCountQis = y.getValue2()

        For each φ ∈ functions deriving from yState:
                If F(yState, φ) ∈ Qi, then:
                        stateCountQis[i] += 1

                If there exists i, 0 <= i <= length(Qi) such that
stateCountQis[i] = k – length(Qi') + 1, then:
```

```
                                        Vq = Vq ∪
{...yFunctionSequence, φ}
                            Else:
                                        Y = Y ∪
Triplet({...yFunctionSequence, φ}, F(yState, φ),
updatedStateCountQis)

Return Vq
```

E. *Algorithm for computing the test suite U*

This algorithm is designed to derive the test suite U using the results obtained from algorithm *d*.

```
Input: k ← estimated number of states in the IUT
Output: SetOfInputSequences (U)
-------------------------------------------------------------------------------

U = {}

For each q ∈ Qr:
        pq ← Get realisable sequence for state q
        Vq = generateVq(k, q) ← Algorithm f

        For each t ∈ Vq:
                s = pq tWs
                U = U ∪ s

Return U
```

5.3. *Correctness of an SXM description using the T-SXM tool*

The T-SXM tool supports two basic levels of correctness for an SXM specification, namely the *semantic* level, and the *logical* level.

A. *Semantic level*

This level guarantees that an SXM specification is at least mathematically correct. At least three basic conditions should be checked:

- *Data types*: the types contain the parameters and returning values that is legally accepted by the specification. Data type checking is also applied for the definition of external operators. They are used to define the types of the input, output and memory. Consequently, the system is responsible for ensuring that the values returned by the operators are of the expected data type.
- *Sets*: anything that is defined as a set (i.e., data type set, state set, function set, transition set) should not contain duplicated elements.
- *Functions*: each function is applied to a tuple containing an input and a memory state, and produces another tuple containing an output and a memory state. Any typical Java function that has the *OutputMemoryPair* object as the return type and takes the *InputMemoryPair* object as the only parameter will automatically be treated as a processing function. Therefore, the system guarantees that the same function should not be able to produce more than one output – memory state tuples for one input – memory state tuple. Note that if $m \in M$, $\sigma \in \Sigma$, and $(m, \sigma) \in dom\ \varphi$, then φ must return a **non-null** *OutputMemoryPair* object.

B. *Logical level*

This level takes care of the checking for the model's general consistency. Questions referring to the model's *input-completeness*, *output-distinguishability*, *minimality,* etc. should be addressed. These conditions must be checked before starting the test generation process.

6. CASE STUDY

A patient getting diabetic has four stages namely pre-metabolic syndrome, metabolic syndrome, pre-diabetic, and type II diabetes. A patient can start off with any single stage of these stages, depending on the time of the diagnosis.

To begin with, we identify the possible paths that the disease can take and look for all the possible scenarios that a patient can progress with the disease. The stage of the disease is initially determined by evaluating the number of positive symptoms that a patient is experiencing at the time of the diagnosis. The evaluation criteria of the **symptoms** are as follows.

- a waist circumference of:

  - 94cm or more in European men, or 90cm or more in South Asian men,
  - a waist circumference of 80cm or more in European and South Asian women.
- high triglyceride levels (fat in the blood) and low levels of HDL (the "good" cholesterol) in the blood, which can lead to atherosclerosis (where arteries become clogged with fatty substances such as cholesterol).
- high blood pressure (BP) that's consistently 140/90mmHg or higher.
- an inability to control blood sugar levels (insulin resistance).
- a tendency to develop irritation and swelling of body tissue (inflammation).

Based on the preliminary identification of number of symptoms, medical practitioners can determine which state of the disease a patient is in. With a set of determination criteria in Table 1, we designed the state-transition diagram (Fig. 2) for Stream X-Machine that represents the full diagnostic process. The first four states (*Start, Triglyceride levels checked, Waist circumference checked,* and *BP checked*) help to count the number of symptoms the patient is suffering from. The five states that proceed to the diagnostics of the disease are *Normal*, *Pre-Metabolic Syndrome*, *Metabolic Syndrome*, *Pre-Diabetes*, and *Type II Diabetes*. These five states are relevant to the data gathered in the perspective of the symptoms. We consider that the initial state of diabetes is *Normal*. If a patient is presenting with one or two symptoms, they are in the *Pre-Metabolic Syndrome* state. Thereafter, if a patient presents with three or more symptoms, they are in the *Metabolic syndrome* state. Likewise, if the patient is presenting with four or more symptoms, they are in *Pre-diabetes* state. Finally, a patient is in *Type II diabetes* state if they have five or more symptoms.

Table 1. Diabetic state determination criteria

Diabetic state	**Number of symptoms that a patient presents at the time of diagnosis**
Pre-metabolic syndrome	One to two
Metabolic syndrome	Three or more
Pre-diabetic	Four or more
Type II diabetic	Uncontrollable insulin levels

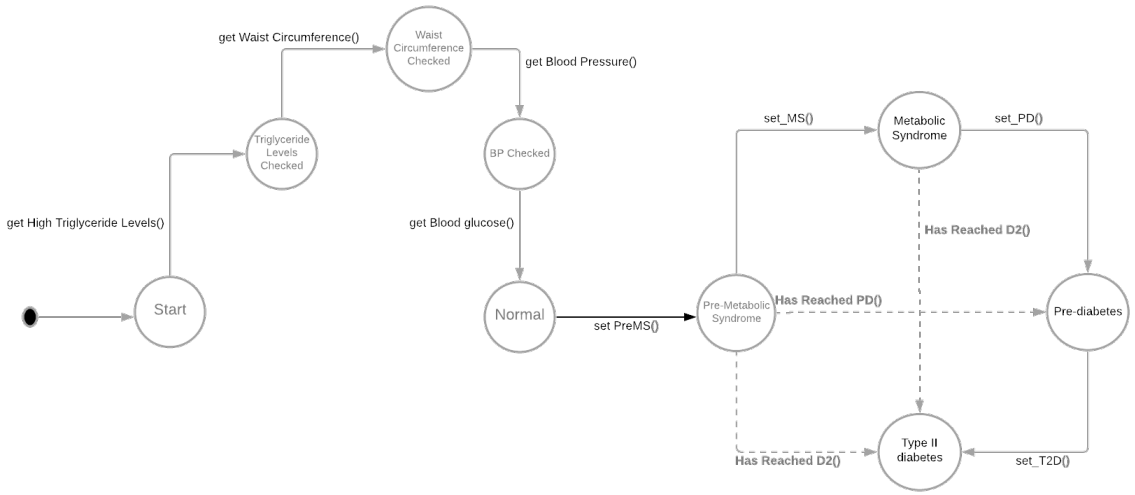

Fig. 2. State-transition diagram for Type 2 diabetes diagnostic process

7. RESULTS

Let n and n' be the number of states of the specification and the IUT, respectively, and k = number of processing functions in the specification. The worst scenario is when $S_r = W_s = \{\varepsilon\}$, which means only the initial state is r-reachable by S_r and no states are pairwise separated by W_s, so the Q_i set only contains singletons. As a consequence, the test set consists of all sequences that reach some states in Q n' times. According to [22], as there are n states in Q, the upper bound on the number of such sequences is proportional to $k^{n.n'}$. However, this extreme seldom happens in practice. In usual applications, all states are r-reachable and there are at most only a few pairs of states that are non-separable. In most scenarios, it is safely to assume that the number of states in the specification represents a good approximation for the number of states in the IUT (otherwise, it would demonstrate a gross misunderstanding from the part of the developers) [22]. As a result, in the experiment, we do not assume n' to be significantly larger than n. Also, the test generation was performed on a 3.1 GHz Dual-Core Intel Core i5.

Table 2 shows the number of test cases generated and the time required for the test generation for the Type II diabetes case study discussed in section 6 with different values of n'. The absolute times in seconds are provided only as an indication.

Table 2. Number of generated test cases and generation time for different values of n'

n'	10	11	12
Number of test cases	5296	5414	5526
Time to generate (sec)	1.313	1.602	1.343

8. RELATED TOOLS

Currently, a small number of SXM tools is available. Most tools concentrated on the modelling and animation of SXM models like [33] and [34].

SXMtool [35] is a tool for SXM models and the automatic test generation. However, it is not evident if this tool is capable of generating concrete test cases, since the authors only demonstrated the generation of the processing function sequences.

JSXM [17] is a tool that supports the animation of SXM models for model validation, the automatic generation of abstract test cases from the SXM specifications and the transformation of abstract test cases into concrete test cases in the implementation language of the SUT. This tool is supposed to be the only tool available for automatically generating concrete (executable) test cases based on the SXM testing method.

In comparison, although T-SXM does not support model animation like the other tools, it can generate concrete test cases for an SXM specification just like what JSXM is capable of. However, T-SXM overcomes JSXM with the ability to validate an SXM specification (e.g., check for accessibility, check for the output-distinguishability condition, etc.). Also, while JSXM requires manual input of the r-state cover to start generating test cases, T-SXM can automatically detect all the r-reachable states and construct the corresponding r-state cover. To the best of our knowledge, T-SXM is the only tool that uses Java as the language for specifying an SXM specification using intuitive wrapper classes. This is considered as an advantage for T-SXM over the other tools as the processing functions can be implemented in a much more intuitive way compared to using in-line C or Java code within XMML or XMDL.

9. CONCLUSIONS

In conclusion, given that this study is based on the theoretical foundations presented in [10], the newly proposed T-SXM tool is built based on the state-counting approach which helps to

overcome the limitations that have been presented in the previous version. In order to demonstrate the improvements of the current version, we have employed a case study from the medical domain where it is very challenging to design models which adhere to the design-for-test conditions. With the success of modelling and generating test sequences for a chronic disease (Type II diabetes), this paper has demonstrated that this tool can be employed not only in the software engineering domain but also in the other domains where dynamic and complex models are required. As the main researchers of this project, we believe that this is a significant milestone not only in improving the T-SXM tool, but also in expanding the horizons of its capabilities of being employed in other domains. We will conduct further research to improve the tool and make it more effective for testing object-oriented systems, especially those written in the Java programming language.

REFERENCES

[1] B. M. Sörensen, A. J. H. M. Houben, T. T. J. M. Berendschot, J. S. A. G. Schouten, A. A. Kroon, C. J. H. van der Kallen, R. M. A. Henry, A. Koster, S. J. S. Sep, P. C. Dagnelie, N. C. Schaper and M. Schram, "Prediabetes and Type 2 Diabetes Are Associated With Generalized Microvascular Dysfunction," *Circulation,* vol. 134, no. 18, pp. 1339-1352, 2016.

[2] J. . F. Ferguson, H. Allayee, R. E. Gerszten, F. Ideraabdullah, P. M. Kris-Etherton, J. M. Ordovás, E. B. Rimm, T. J. Wang and B. J. Bennett, "Nutrigenomics, the Microbiome, and Gene-Environment Interactions: New Directions in Cardiovascular Disease Research, Prevention, and Treatment," *Circulation: Cardiovascular Genetics,* vol. 9, no. 3, pp. 291-313, 2016.

[3] P. J. Meikle, G. Wong, C. K. Barlow, J. M. Weir, M. A. Greeve, G. L. MacIntosh, L. Almasy, A. G. Comuzzie, M. C. Mahaney, A. Kowalczyk, I. Haviv, N. Grantham, D. J. Magliano and J, "Plasma Lipid Profiling Shows Similar Associations with Prediabetes and Type 2 Diabetes," *PLoS ONE,* vol. 8, no. 9, p. e74341, 2013.

[4] A. Sigruener, M. E. Kleber, S. Heimerl, G. Liebisch, G. Schmitz and W. Maerz, "Glycerophospholipid and sphingolipid species and mortality: the Ludwigshafen Risk and Cardiovascular Health (LURIC) study," *PLoS One,* vol. 9, no. 1, p. e85724, 2014.

[5] J. Basu, R. Avila and R. Ricciardi, "Hospital Readmission Rates in U.S. States: Are Readmissions Higher Where More Patients with Multiple Chronic Conditions Cluster?," *Health Services Research,* vol. 51, no. 3, pp. 1135-1151, 2015.

[6] C. Buttorff, T. Ruder and M. Bauman, Multiple Chronic Conditions in the United States, Santa Monica, CA: RAND Corporation, 2017.

[7] O. Ojo, "Nutrition and Chronic Conditions," *Nutrients,* vol. 11, no. 2, p. 459, 2019.

[8] E. I. Mandel, E. N. Taylor and G. C. Curhan, "Dietary and Lifestyle Factors and Medical Conditions Associated with Urinary Citrate Excretion," *Clinical Journal of the American Society of Nephrology,* vol. 8, no. 6, pp. 901-908, 2013.

[9] K. M. Kolasa and K. Rickett, "Barriers to Providing Nutrition Counseling Cited by Physicians," *Nutrition in Clinical Practice,* vol. 25, no. 5, pp. 502-509, 2010.

[10] M. Flynn, C. Sciamanna and K. Vigilante, "Inadequate physician knowledge of the effects of diet on blood lipids and lipoproteins.," *Nutrition Journal,* vol. 2, no. 1, 2003.

[11] M. A. Schaub, T. A. Henzinger and J. Fisher, "Qualitative networks: a symbolic approach to analyze biological signaling networks," *BMC Systems Biology,* vol. 1, no. 1, pp. 1-21, 2007.

[12] A. Naldi, D. Thieffry and C. Chaouiya, "Decision Diagrams for the Representation and Analysis of Logical Models of Genetic Networks," in *Computational Methods in Systems Biology*, Berlin, Heidelberg, 2007.

[13] N. Miskov-Zivanov, P. Wei and C. S. C. Loh, "THiMED: Time in hierarchical model extraction and design," *International Conference on Computational Methods in Systems Biology,* pp. 260-263, 2014.

[14] C. Hajat and E. Stein, "The global burden of multiple chronic conditions: A narrative review," *Preventive Medicine Reports,* vol. 12, pp. 284-293, 2018.

[15] I. Stamatopoulou, M. Gheorghe and P. Kefalas, Modelling Dynamic Organization of Biology-Inspired Multi-agent Systems with Communicating X-Machines and Population P Systems, 2005, pp. 389-403.

[16] S. Eilenberg, "Automata, languages and machines," *Academic Press,* vol. A, 1974.

[17] D. Dranidis, K. Bratanis and F. Ipate, "JSXM: A tool for automated test generation," *International Conference on*

Software Engineering and Formal Methods, pp. 352-366, 2012.

[18] M. Holcombe and F. Ipate, Correct Systems: Building a Business Process Solution, Berlin: Springer, 1998.

[19] F. Ipate and M. Holcombe, "An integration testing method that is proved to find all faults," *Internat. J. Comput. Math,* vol. 63, pp. 159-178, 1997.

[20] F. Ipate and M. Holcombe, "Generating test sequences from non-deterministic generalized stream X-machines," *Formal Aspects of Comput.,* vol. 12, no. 6, pp. 443-458, 2000.

[21] F. Ipate and M. Holcombe, "Testing conditions for communicating stream X-machine systems," *Formal Aspects of Comput.,* vol. 13, no. 6, pp. 431-446, 2002.

[22] F. Ipate and D. Dranidis, "A unified integration and component testing approach from deterministic stream X-machine specifications," *Formal Aspects of Computing,* vol. 28, no. 1, pp. 1-20, 2016.

[23] K. Bogdanov, M. Holcombe, F. Ipate, L. Seed and S. Vanak, "Testing methods for X-machines, a review," *Formal Aspects of Comput.,* 2006.

[24] F. Ipate, "Testing against a non-controllable stream X-machine using state counting," *Theoretical computer science,* vol. 353, pp. 291-316, 2006.

[25] K. Phung and E. Ogunshile, "An algorithm for implementing a minimal stream X-Machine model to test the correctness of a system," *2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT),* pp. 93-101, 2020.

[26] M. A. Boemo, L. Cardelli and C. A. Nieduszynski, "The Beacon Calculus: A formal method for the flexible and concise modelling of biological," *PLOS Computational Biology,* vol. 16, no. 3, 2020.

[27] R. Adnan, H. Osman, S. Umair and T. Sofiène, "Formal reasoning about systems biology using theorem proving," *PLOS ONE,* vol. 12, pp. 1-27, 2017.

[28] Q. Wang and E. M. Clarke, "Formal modeling of biological systems," *2016 IEEE International High Level Design Validation and Test Workshop (HLDVT),* pp. 178-184, 2016.

[29] J. Mortimer (Ed.), "The FMS Report: Ingersoll Engineers," IFS Publications, Kempston, Bedfordshire, 1984.

[30] S. Wolfram, A New Kind of Science, Wolfram Media, 2002.

[31] F. Ipate and M. Holcombe, "A method for refining and testing generalized machine specifications," *J. Computer Math,* vol. 68, pp. 197-219, 1998.

[32] F. Ipate and M. Holcombe, "Testing data processing-oriented systems from stream X-machine models," *Theoretical Computer Science,* Vols. 2-3, no. 176-191, p. 403, 2008.

[33] P. Kapeti and P. Kefalas, "A design language and tool for X-machines specification," *Advances in Informatics,* 2000.

[34] M. Holcombe, S. Coakley and R. Smallwood, "A general framework for agent-based modelling of complex systems," *European Conf. on Complex Systems,* 2006.

[35] C. Ma, J. Wu and T. Zhang, "Sxmtool: A tool for stream x-machine testing," *World Congress on Software Engineering,* 2010.

[36] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE transactions on software engineering,* vol. 3, pp. 178-187, 1978.

[37] M. Buysschaert, J.-L. Medina, B. Buysschaert and M. Bergman, "Definitions (and Current Controversies) of Diabetes and Prediabetes," *Current Diabetes Reviews,* vol. 12, no. 1, pp. 8-13, 2016.