Journal of Intelligent Learning Systems and Applications, 2015, 7, 21-36
Published Online February 2015 in SciRes. http://www.scirp.org/journal/jilsa
http://dx.doi.org/10.4236/jilsa.2015.71003

Applying DNA Computation to Error Detection Problem in Rule-Based Systems

Behrouz Madahian[1], Amin Salighehdar[2], Reza Amini[3]

[1]Department of Mathematical Sciences, University of Memphis, Memphis, USA
[2]Financial Engineering Department, Stevens Institute of Technology, Hoboken, USA
[3]Department of Computer Science, Florida International University, Miami, USA
Email: bmdahian@memphis.edu, asalighe@stevens.edu, ramin001@fiu.edu

Received 25 January 2015; accepted 10 February 2015; published 13 February 2015

Abstract

As rule-based systems (RBS) technology gains wider acceptance, the need to create and maintain large knowledge bases will assume greater importance. Demonstrating a rule base to be free from error remains one of the obstacles to the adoption of this technology. In the past several years, a vast body of research has been carried out in developing various graphical techniques such as utilizing Petri Nets to analyze structural errors in rule-based systems, which utilize propositional logic. Four typical errors in rule-based systems are redundancy, circularity, incompleteness, and inconsistency. Recently, a DNA-based computing approach to detect these errors has been proposed. That paper presents algorithms which are able to detect structural errors just for special cases. For a rule base, which contains multiple starting nodes and goal nodes, structural errors are not removed correctly by utilizing the algorithms proposed in that paper and algorithms lack generality. In this study algorithms mainly based on Adleman's operations, which are able to detect structural errors, in any form that they may arise in rule base, are presented. The potential of applying our algorithm is auspicious giving the operational time complexity of O(n∗(Max{q, K, z})), in which n is the number of fact clauses; q is the number of rules in the longest inference chain; K is the number of tubes containing antecedents which are comprised of distinct number of starting nodes; and z denotes the maximum number of distinct antecedents comprised of the same number of starting nodes.

Keywords

DNA Computing, Rule-Based Systems, Rule Verification, Structural Errors

1. Introduction

Adoption of expert systems in real world applications has been greatly increased. In past years, much effort has

How to cite this paper: Madahian, B., Salighehdar, A. and Amini, R. (2015) Applying DNA Computation to Error Detection Problem in Rule-Based Systems. *Journal of Intelligent Learning Systems and Applications*, **7**, 21-36.
http://dx.doi.org/10.4236/jilsa.2015.71003

been devoted to analyze different aspects of rule-based systems such as knowledge representation, reasoning, and verification of rule-based systems [1]-[5]. A rule base, which is the central part of an expert system codifies the knowledge from domain expert in the form of inference rules. Often these inference rules are built into a rule base incrementally over years and subject to frequent refinements. Due to the different and even conflicting views provided by domain experts besides the above construction process, a rule base can contain many structural errors. According to [1]-[4] [6], and several other studies, four typical types of structural errors include inconsistency (conflict rules), incompleteness (missing rules), redundancy (redundant rules), and circularity (circular depending rules). Many different techniques have been developed to detect the above errors in rule-based systems. Earlier work mainly focused on detecting structural errors by checking rules pair-wisely [7]-[9]. Recent work aimed at detecting structural errors caused from applying multiple rules in longer inference chains. The majority of recent verification techniques involve using some graphical notations such as graphs [10]-[12], and Petri nets [2] [3] [6] [13]-[15].

DNA computation has emerged in recent years as an exciting new research field at the intersection of computer science, biology, engineering, and mathematics. There exist two main barriers to the continued development of traditional silicon-based computers [16]. Invention of silicon integrated circuits and advances in miniaturization has led to incredible increases in processors speed and memory access time. However, there is a limit to how far this miniaturization can go. Eventually chip fabrication will hit the wall imposed by the Heisenberg Uncertainty Principle (HUP) [16]. DNA computing based on its complementary characteristics and massive parallelism (when a step is performed in an experiment, the operation is performed in parallel on all molecules in the tube) has the potential to solve complex problems such as NP-Complete ones [16]. The physician Richard Feynman first proposed the idea of using living cells and molecular complexes to construct sub-microscopic computers [17]. After Feynman proposal, there has been an explosion of interest in performing computations at molecular level. Adleman who used DNA strands to solve a directed Hamiltonian path problem, indicated the feasibility of a molecular approach to solve combinatorial problems [18]. Subsequently, by solving satisfiability problem (SAT), Lipton demonstrated the advantage of using the massive parallelism inherent in DNA-based computing.

Authors in [1] proposed algorithms, which utilized DNA computing to render an error free rule base for rule-based systems. The algorithms proposed in [1] lack generality since just for special cases of rule base can work and there are cases that structural errors are not removed correctly by utilizing their algorithms. Rules in a Rule-Based system are typically formed as $X \rightarrow Y$, in which X is an antecedent node (AN) and Y is a conclusion node (CN). Two nodes, atomic and compound, exist in rule bases. We are interested in the problem domain of Horn Clauses, as addressed in [2]-[4], which allow only one conclusion part in each rule and compound antecedents in rules are only presented in the conjunction format. So, before we transform rules to their corresponding rule paths, a normalization should be carried out in order to obtain Horn Clause form of the rules [3] [4]. In this paper, we are interested in finding structural errors and the set of rules causing these errors. The reasons of structural errors may be due to rule conflicting, mismatched condition and conclusion, and circular and redundant rules [3] [13] [15]. Inconsistent rules result in conflict, which is the direct source of incorrect rule derivation. Redundant rules increase the size of rule base and cause non necessary reasoning. Incomplete rules prohibit the rule base from activating certain normal rule derivation. Circular dependent rules will force the rule base to run into an infinite loop of reasoning.

In this study, algorithms are developed to cope with all cases. Our algorithms have the ability to detect structural errors in any form that they may occur in rule base. As a result, DNA computing as an alternative to verify structural errors in rule-based systems gains more generality. The remainder of this paper is organized as follows. Structural errors are briefly described in Section 2. In Section 3, we outline DNA computation and introduce our DNA-based algorithms to detect structural errors in Rule-Based systems. We analyze the complexity of our algorithm and conclusion is represented in Section 4.

2. Typical Structural Errors in Rule Bases

- Redundancy. When unnecessary rules exist in the rule base, redundancy occurs. These rules not only increase the size of the rule base but also may cause additional useless inferences. Redundancy is a potential source of inconsistency when knowledge is updated [4]. A rule is redundant with respect to the conclusion if two rules have identical conditions and conclusions (identical rules) or two rules have identical conclusions while the

condition for one rule is either a generalization or special case of the condition for the other one [3] [4]. The rule, which has more general condition subsumes (is stronger than) the other rule. Logical redundancy implies operational redundancy [4]. Thus, subsumed (less general) rules can be eliminated, which has no influence on logical inference capability [4].

- Incompleteness. When there are missing rules in a rule base, incompleteness occurs. Except the rules for representing facts and goal nodes, a rule is called as a useless rule if the rule's condition (conclusion) cannot be matched by other rules' conclusion (condition). The unmatched condition are called dangling conditions, while the unmatched conclusions are called dead-end conclusions. Mostly, the reasons of useless rules are due to some missing rules.
- Circularity. When two or several rules have circular dependency, Circularity occurs. Circular dependent rules can cause infinite reasoning and must be broken.
- Inconsistency. Since inconsistent rules result in conflict facts, for correct functioning of an expert system, inconsistency must be resolved. Two rules r_1 and r_2 (that their conclusions are not compatible) are inconsistent if there exists a state, such that simultaneously both antecedents (pre-conditions) of r_1 and r_2 can be fired [4].

2.1. The Structure of DNA and Basic Denaturing and Annealing Operations

DNA (deoxyribonucleic acid) encodes the genetic information of cellular organisms [16]. It consists of polymer chains, commonly referred to as DNA strands. Each strand may be viewed as a chain of nucleotides, or bases, attached to a sugar-phosphate backbone. The four DNA nucleotides are Adenine, Guanine, Cytosine, and Thymine, commonly abbreviated to "A", "G", "C", and "T" respectively. Each strand, according to chemical convention, has a 5′ and 3′ end. Thus, any single strand has a natural orientation. This orientation is due to the fact that one end of the single strand has a free 5′ phosphate group, and the other end has a free 3′ deoxyribose hydroxyl group. "A" bonds with "T" and "G" bond with "C". The pairs (A, T) and (G, C) are therefore known as complementary base pairs. The two pairs of bases form hydrogen bonds between each other. Double stranded DNA may be dissolved into single strands (denatured), by heating the solution to a temperature determined by the composition of the strand [19]. Heating breaks the hydrogen bonds between complementary strands. Annealing is reverse of denaturing, whereby a solution of single strands is cooled, allowing complementary strands to bind together (**Figure 1**).

Figure 1 demonstrate the annealing of 5′ end of a single strand DNA to the 3′ end of another DNA in presence of DNA ligase and a splint. In this figure splint has 20 base pairs and consists of the complement of the 10 nucleotides at the 3′ end of one strand and the complement of 10 nucleotides at the 5′ end of the other.

In order to anneal the 5′ end of a single strand DNA to the 3′ end of another DNA, in the presence of "DNA ligase" we hybridize a set of specific splint oligos of length 20. Each splint consists of the complement of the 10 nucleotides at the 3′ end of one strand and the complement of 10 nucleotides at the 5′ end of the other.

2.2. Initial Set Construction

Oligonucleotides uniquely encoding each node and splint are assigned. As proposed by Barich [20], there are

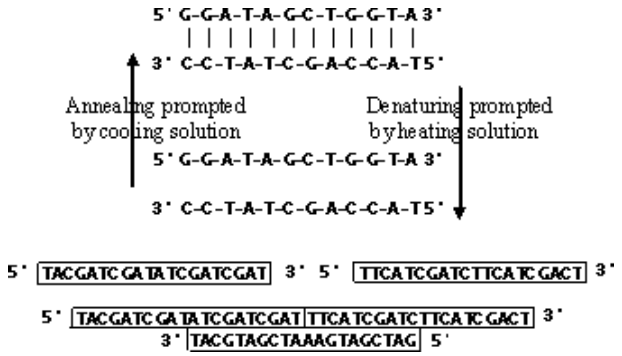


Figure 1. Annealing and denaturing and using annealing to form a long single stranded DNA.

some constraints for strand library design indicating that sequences must be designed in a way that strands have little secondary structure in order to prevent unintended probe-library hybridization. Thus, short oligonucleotides uniquely encoding each node and splint must be used. Then, splints are used to join to their complement sequence. Hence, by defined polarity, short single stranded oligos which represent nodes covalently join and create longer single stranded DNA molecules. The above procedure enables us to encode each rule path, including appropriate nodes in form of long single stranded DNA.

2.3. Operations Description

We use the basic operations on strands that are defined by Adleman [18], the Parallel Filtering Model of Amos [16], and an alternative filtering-style model called the Stickers Model developed by Roweis [21]. In all filtering models a computation consists of a sequence of operations on finite multi-set of strands. It is normally the case that a computation begins and terminates with a single multi set of strands. An initial solution consists of strands which are of length O(n). Where n is the problem size. The initial solution should include all possible solutions (Each encoded by a strand) to the problem to be solved. The point here is that the initial set in any implementation of the model is assumed to be relatively easy to generate as a starting point for the computation. The computation then proceeds by filtering out strands which encode illegal solution and cannot be the result. The operations defined in parallel filtering models and Adleman experiments are as follows. The implementation of these operations can be found in detail in [16] [18] [21] and [22].

- Separate (T, s, k, T_{on}, T_{off}): This operation separates strands that contain sequence "s" starting from position "k", into T_{on}; otherwise, into T_{off} [21].
- Extract (T, s, T^+, T^-): Given a tube T and a sub-strand "s", this operation creates two new sets T^+, T^-, where T^+ include all strands in T containing "s", and T^- includes all strands in T that do not contain "s" [18].
- Union (T, T_1, T_2, …, T_n): This operation creates set T, which is the set union of the T_1, T_2, …, T_n [16].
- Copy (T, T_1, T_2, …, T_n): This operation produces copies T_1, T_2, …, T_n of the set T [16].
- Detect (T): Given a set T, this operation returns true (Y), if T contains at least one DNA strand; Otherwise, it returns false (N) [22].
- Read (T): This operation describes each DNA strand in set T [22].
- Remove (T): This operation removes all strands in tube T [22].

2.4. Encoding Inference Rules by DNA Strands

For a general inference rule with compound antecedent ($R:(X_1 \Lambda \ldots \Lambda X_n) \rightarrow Y$), each antecedent node (AN) and conclusion node (CN) are encoded by a 20-mer DNA strand. To encode the relations "Λ" and "→", two tetra-nucleotide sequences, "AAAA" and "CCCC" (and their complements) are used respectively. Thus, by creating 24 nucleotide long splints, which contain the appropriate tetra-nucleotide, relations "$X_i \Lambda X_j$" and "$X_i \rightarrow X_j$" are enforced [1]. The "$X_i \rightarrow X_j$" is a splint whose sequence in the 3′-5′ direction is the concatenation of the complement of 10 nucleotides at the 3′-end of the strand node X_i, four nucleotides of "CCCC", and the complement of 10 nucleotides at the 5′-end of the strand X_j. Similarly, "$X_i \Lambda X_j$" is a splint whose sequence is the concatenation of complement of 10 nucleotides at the 3′-end of the strand X_i, four nucleotides of "AAAA", and the complement of 10 nucleotides at the 5′-end of strand X_j. All strands representing starting nodes are designed in the way that, all of them have common sub-strand "TTTTTTTTTT" at the 5′ end of their strands and all strands representing the goal nodes are designed in the way that all of them contain the common sub-strand "GGGGGG GGGG" at the 3′ end of their strands. Thus, sequences "AAAAAAAAAA" and "CCCCCCCCCC" are needed in our algorithm to distinguish these nodes, as explained in Section 3.6.1. Finally, in order to make sure that strands representing the complements of "TTTTTTTTTT" and "GGGGGGGGGG" will bond to the starting nodes and goal nodes respectively and nowhere else, all strands representing the other nodes should be designed in the way that they do not have successive "G"s or "T"s at the 3′ or 5′ end of their strands. For instance **Figure 2(a)** shows three distinct strands representing nodes. **Figure 2(b)** shows two distinct strands representing "Λ" operator and two distinct strands representing "→" operator. The resulting strands are shown in **Figure 2(c)**. Since permutations of k-node compound antecedent creates the same antecedent (k! strands representing the same AN); therefore, for each rule with compound antecedent, one of the CN for the k! strands is chosen randomly and poured into tube T_{r1} in order to detect redundancy and circularity [1]. We encode all permutations of antecedent of the rules with compound AN and pour them into tube T_Λ. In order to detect subsumed rules, special tubes $T_{\Lambda sk}$ are

X_1 : 5′ TACGATCGATATCGATCGAT 3′
X_2 : 5′ TTCATCGATCTTCATCGACT 3′
X_3 : 5′ CACTATATCCTCGCTATTCA 3′

(a)

$X_1 \Lambda X_2$: 3′ TAGCTAGCTAAAAAAAGTAGCTAG 5′
$X_2 \Lambda X_1$: 3′ AAGTAGCTGAAAAAATGCTAGCTA 5′
$X_2 \rightarrow X_3$: 3′ AAGTAGCTGACCCCGTGATATAGG 5′
$X_1 \rightarrow X_3$: 3′ ATGCTAGCTACCCCGTGATATAGG 5′

(a)

$R_1 : (X_1 \Lambda X_2) \rightarrow X_3$
5′ TACGATCGATATCGATCGATTTTTTTCATCGATCTTCATCGACTGGGGCACTATATCCTCGCTATTCA 3′
3′ TAGCTAGCTAAAAAAAGTAGCTAGAAGTAGCTGACCCCGTGATATAGG 5′
$R_2 : (X_2 \Lambda X_1) \rightarrow X_3$
5′ TTCATCGATCTTCATCGACTTTTTTACGATCGATATCGATCGATGGGGCACTATATCCTCGCTATTCA 3′
3′ AAGTAGCTGAAAAAATGCTAGCTAATGCTAGCTACCCCGTGATATAGG 5′

(c)

Figure 2. (a) Distinct strands representing each node; (b) Splints representing "Λ" and "→" operators; (c) The resulting long strands R_1 and R_2, both representing the same rule.

created as described below. The antecedent of all rules with compound AN, which are comprised of k starting nodes, are poured into tube $T_{\Lambda sk}$. The strands are labeled with k_i. For instance, $T_{\Lambda s1}$ is comprised of antecedents of all rules with atomic AN of starting nodes and are labeled as 1_i strands (i^{th} strand of tube $T_{\Lambda s1}$) and $T_{\Lambda s2}$ is comprised of antecedents of all rules with compound AN of two starting node labeled as 2_i strands. Thus tubes $T_{\Lambda sk}{}^c$, Comprised of many copies of complements of strands in tubes $T_{\Lambda sk}$ are created. These are used in Detect Redundancy algorithm described in Section 3.6.4.

2.5. Generating the Solution Space

An essential difficulty in all filtering models is that initial multi sets of strands generally have quantity, which is exponential in the problem size [16]. What is done in practice is that an initial set is constructed containing a polynomial number of distinct strands. The design of these strands ensures that exponentially large initial set of the system (rule paths) can be generated automatically [16]. Sequence of pre-steps are carried out in order to create the initial solution space containing rule paths. An example of creating the initial set for a simple rule base is depicted in **Figure 3**. These pre-steps are alike what is presented in [1], as follows.

Pre-Step 1. Tubes T_Λ, $T \rightarrow$, T_{r1}, T_{r2}, T_{r3}, T_c, and T_s are needed at this stage. Strands representing splints "XΛY" And "X → Y" are poured into tube T_Λ and T→ respectively. Starting nodes and goal nodes are not conclusions and antecedent parts (conditions) of any rule respectively. Thus, in order eliminate these kinds of rules and prevent circularity from occurring to starting nodes (in our sample rule base, X_1) and goal nodes (in our sample rule base, X_6), copies of strands designed as "GGGG" followed by the 10 nucleotides at the 5′ end of starting nodes and 10 nucleotides at 3′ end of the goal nodes followed by "GGGG" are poured into tube T→. Thus any splint in T→ that anneals to the above strands, is removed. We may have rules in which the goal nodes are included in compound antecedents. In order to eliminate these kinds of rules, copies of strands designed as 10 nucleotides at 3′ end of goal nodes followed by "TTTT" and "TTTT" followed by 10 nucleotides at the 5′ end of the goal nodes are poured into tube T_Λ. Thus, any splint in tube T_Λ that anneals to the above strands is eliminated (assuming Y as goal node, splints formed as X_i Λ Y and Y Λ X_i are eliminated). Next the CN of rules with compound AN are poured into tube T_{r1} and strands representing CN for each atomic AN is poured into tube T_{r2}. In order to identify which CN has more than one rule leading to it, strands in T_{r2} are poured into tube T_{r1} to make sure that T_{r1} contains the CN for all rules. Then only one copy of the complements for all CN is poured into T_{r1}. Any CN in T_{r1}that does not anneal to its complement represents CN with more than one rule leading to it and is poured intoTr3. Next copies of all AN are poured into T_c.

Pre-Step 2. Splints in T_Λ and copies of strands "TTTT" are poured into tube T_c and DNA ligation is allowed to occur. By means of splints, each set of compound AN would stick together and creates a double stranded DNA in length of the splint. Then single strands are separated from T_c to T_s.

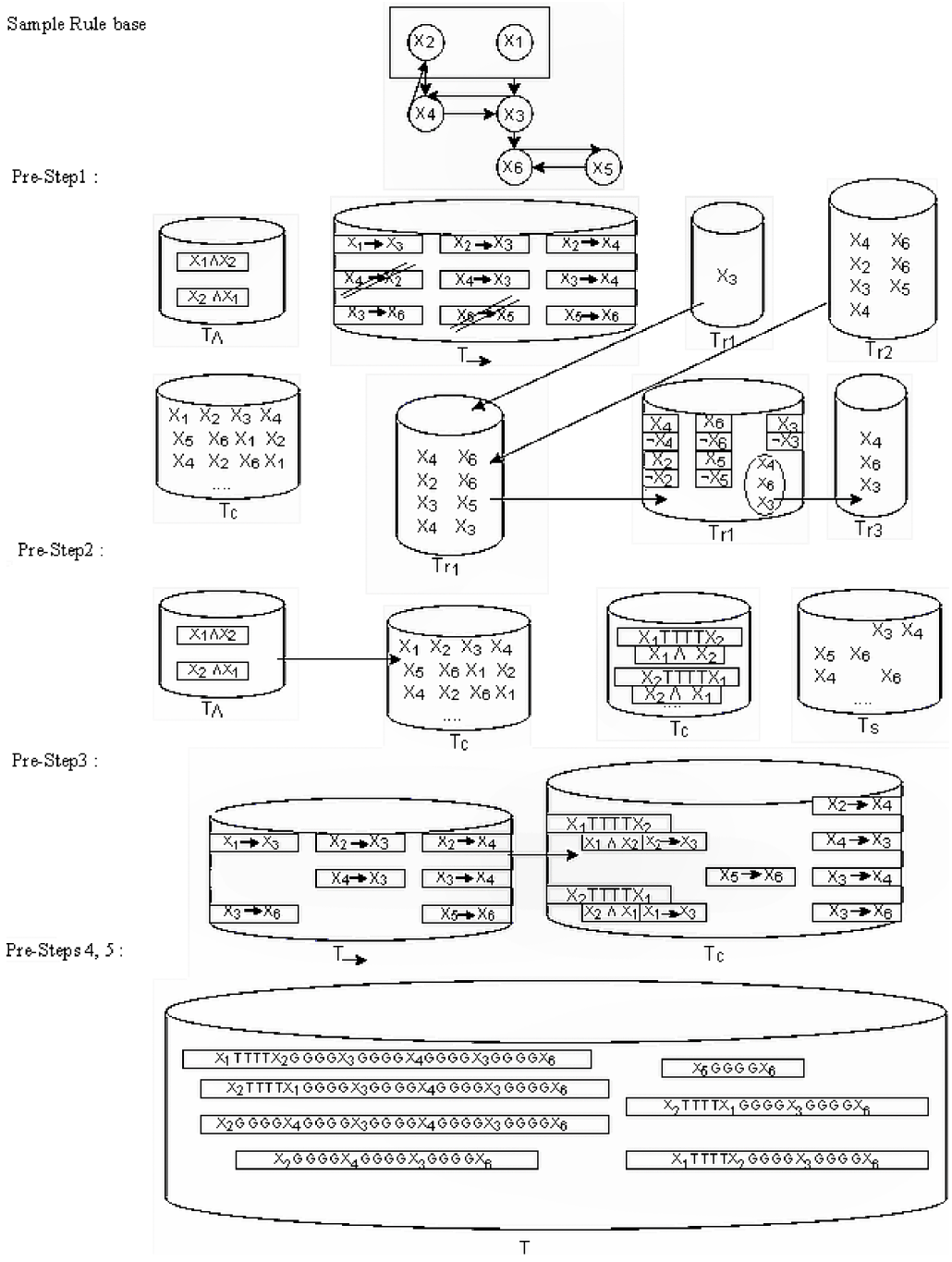


Figure 3. Initial set generation.

Pre-Step 3. To process the situation that one of the compound AN of a rule is the CN of another rule, population of strands from T→ are poured into tube T_c. Splints in T→ bond to the strands mentioned above and long double stranded DNAs, which are subset of rule chains are formed.

Pre-Step 4. At this stage copies of "GGGG" and the strands from T_s are poured into T_c to form rule-chain subsets containing atomic AN. Each long strand created at this step corresponds to one set of possible inference rule paths that may contain any of typical structural errors.

Pre-Step 5. At this stage, double-stranded DNAs from T_c are denatured and poured into tube T. Finally, all possible rule sequences are represented in T.

2.6. Detection of Structural Errors

2.6.1. Removing Incomplete Rule Paths

In order to remove incomplete rule paths, which do not start with starting nodes or do not lead to the goal node the algorithm below is performed [1].

Detect completeness algorithm

1) Input (T);
2) Extract (T, "TTTTTTTTTT", T_y, T_{incomp});
3) Extract (T_y, "GGGGGGGGGG", T, T_{incomp});
4) Remove (T_{incomp}).

All strands containing at least one starting node in their sequence are extracted from T into tube T_y; otherwise, into tube T_{incomp} at line 2 (multiple starting nodes can be at the beginning of rule paths in form of compound antecedent of the first rule). This extract operation is carried out by pouring many copies of strand "AAAAAAAA AA" into tube T. This strand only anneals to single strands containing "TTTTTTTTTT". As explained in Section 3.4, only strands representing starting nodes are designed so that all of them have this sub-strand. At line 3, strands containing goal nodes are extracted from T_y and poured into tube T; otherwise, into tube T_{incomp}. This extract operation is carried out by pouring many copies of strand "CCCCCCCCCC" into tube T. This strand only anneals to single strands containing "GGGGGGGGGG". As explained in Section 3.4, only strands representing goal nodes are designed so that all of them have this sub-strand. At the end of algorithm, strands in tube T_{incomp} represent incomplete rule paths and should be removed. By performing this algorithm just one time, all complete rule-paths with different starting nodes and goal nodes are extracted. Thus, there is no need to perform the algorithm for all starting nodes and goal nodes repeatedly. Assuming that X_i is a starting node and Y_i is a goal node, it should be noted that there is no splint to complement the 10 nucleotides at the 5′ end of X_i; therefore, in all strands containing X_i, it has to be located in front of the strands (in case of starting nodes in form of compound AN there is no splint to complement 10 nucleotides at the 5′ end of first starting node). Similarly, there is no splint to complement the 10 nucleotides at the 3′ end of Y_i. Thus, in all strands containing Y_i, it has to be located at the end of the strand.

2.6.2. General Algorithm to Detect Circularity

Algorithm proposed in [1] is aimed at removing all the circularity depending rules that may exist, by removing strands, in which the same node appears in at least two location. Only strands containing the nodes that exist in T_{r3} are likely to have circularity problem. Assuming z to represent the number of nodes in T_{r3}. We modify the algorithm presented in [1] and call it "Detect Circularity Part 0" algorithm as follows.

Detect Circularity Part 0

1. Input (T, T_{r3}, X_i), for each node X_i in T_{r3}
2. Copy (T,..., T_i,…,T_z)
3. q=1
4. Do while Detect (T_i)=(Y)
5. Separate (T_i, GGGGGGGGGG, (q*24), T_i^f, T_i)
6. In Parallel — Separate (T_i, X_i, (q*24), T_i^{tmp}, T_i
7. Separate (T_i^+, X_i, (q*24), T_i^{cir}, T_i^+)
8. Union (T_i^+, T_i^{tmp})
9. q=q+1
10. End do
11. Remove (….T_i,…, T_Z)

Tube T_i^f is composed of strands containing any goal node at position q∗24 and thus there is no need to be compared. At the end of this algorithm, all rule chains, in which node X_i appears at least twice in the strand are poured into tubes T_i^{cir}. We remove these strands from T. There are some cases that this algorithm is unable to remove circularity error and after applying the algorithm, circularity error will remain in rule base. Suppose that after performing above algorithm, X_i and X_j are found to be circular nodes and there exist at least two paths between nodes X_i and X_j or more precisely there exist two rules or chains of rules acting reverse between these two nodes (e.g. $X_i \rightarrow X_j$ and $X_j \rightarrow X_i$) besides one or more distinct chains of rules from a starting node (starting

nodes can be distinct) leading to nodes X_i and X_j in addition to distinct chains of rules from these nodes leading to the goal node. In such a situation, this algorithm is unable to remove circularity error. That is, by removing rule paths which have at least two occurrences of nodes X_i and X_j, circularity error will not be removed. In order to clarify these situations, take simple rule base shown in **Figure 4** as an example. Assuming X_1 and X_4 are starting and goal nodes respectively. The resulting directed graph made by these rules and all the paths starting from node X_1 and leading to X_4 is depicted in **Figure 4**.

By performing the "Detect Circularity Part 0" for nodes X_2 and X_3 strands number 5 and 6 are detected to have two occurrence of these nodes respectively. These strands are poured into tubes T_1^{cir} and T_2^{cir}. Next, we remove these strands from T. Now, if we establish the directed graph made by rules embedded in remainder paths, we see that removing paths 4 and 5 will not result in elimination of any of rules causing circularity error. That is, these rules (R_4: $X_2 \rightarrow X_3$, R_5: $X_3 \rightarrow X_2$) exist in other rule paths. Thus, this error remains and the algorithm is unable to remove it. As a matter of fact this is the case for all rule base with rules or chains of rules acting reverse between circular nodes, in addition to rules or chains of rules from a starting node leading to each one of these circular nodes and having from each circular node, paths or more precisely chains of rules, leading to the goal node. Thus, by means of these algorithm circularity error for these cases cannot be eliminated. To make these statements more clear, another instance of rule base and its corresponding directed graph is depicted in **Figure 5**. Assume nodes X_1, X_5 to be starting node and goal node respectively. As it is obvious from the directed graph of this rule base, there exists a circle in this rule base comprised of rules R_5, R_6, and R_7. These rules are circularly dependent. Similar to the previous section we make an attempt to remove this error by means of above algorithm. All the complete paths start from node X_1 leading to X_5 are as follows.

By executing the algorithm that explained above, for nodes, which are present in T_{r3} (X_4, X_3, and X_4), circular paths {4, 8, 12} are found in which nodes X_2, X_3, and X_4 appear twice in these strands respectively. We take into account the remainder paths and establish the directed graph constructed by rules embedded in them (rule paths {1, 2, 3, 5, 6, 7, 9, 10, 11}). Obviously the directed graph made by these rules is the same as the directed graph of original rule base. We notice that all circularly dependent rules (R_5, R_6, R_7) still exist in remainder paths (and consequently in rule base) and none of them is removed. Thus, after performing the algorithm, circularity error still exists in rule base. It should be noted that, as it is obvious from our examples, in such a situation there exist complete rule paths having two occurrence of circular node X_i, in which circular node X_j is located between two position of X_i. And there exist complete rule paths having two occurrence of circular node X_j in which circular node X_i is located between two position of X_j (in our example paths 4, 8, and 12 have this property for nodes X_4, X_3, and X_4).

In this section we propose an algorithm, which can perfectly remove circularity errors. This algorithm comprises three parts performed for each pair from circular nodes (X_i, X_j) that has been found in the previous

1: $X_1 \rightarrow X_2 \rightarrow X_4$
2: $X_1 \rightarrow X_3 \rightarrow X_4$
3: $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4$
4: $X_1 \rightarrow X_3 \rightarrow X_2 \rightarrow X_4$
5: $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_2 \rightarrow X_4$
6: $X_1 \rightarrow X_3 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4$

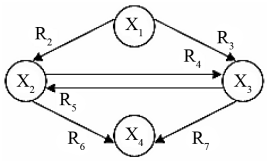


Figure 4. Rule base with circularity error.

1: R_2R_8: $X_1 \rightarrow X_2 \rightarrow X_5$
2: $R_2R_5R_6R_9$: $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5$
3: $R_2R_5R_{10}$: $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_5$
4: $R_2R_5R_6R_7R_8$: $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_2 \rightarrow X_5$
5: R_3R_9 : $X_1 \rightarrow X_4 \rightarrow X_5$
6: $R_3R_7R_8$: $X_1 \rightarrow X_4 \rightarrow X_2 \rightarrow X_5$
7: $R_3R_7R_5R_{10}$: $X_1 \rightarrow X_4 \rightarrow X_2 \rightarrow X_3 \rightarrow X_5$
8: $R_3R_7R_5R_6R_9$: $X_1 \rightarrow X_4 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5$
9: R_4R_{10} : $X_1 \rightarrow X_3 \rightarrow X_5$
10: $R_4R_6R_9$: $X_1 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5$
11: $R_4R_6R_7R_8$: $X_1 \rightarrow X_3 \rightarrow X_4 \rightarrow X_2 \rightarrow X_5$
12: $R_4R_6R_7R_5R_{10}$: $X_1 \rightarrow X_3 \rightarrow X_4 \rightarrow X_2 \rightarrow X_3 \rightarrow X_5$

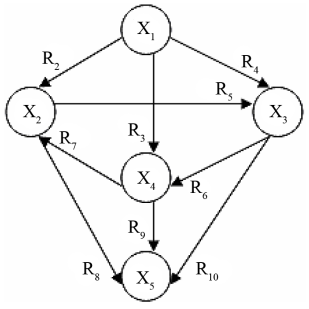


Figure 5. Rule-base with circularity error.

algorithm. It should be noted that all paths at this stage start from starting nodes and there are no rules in which starting nodes are inferred from them. Thus, in performing different parts of our Detect Circularity algorithm, we don't need to check position 0 to 24 of the paths. That is, in all parts of Detect Circularity algorithm, q is initially equal to one (q = 1). First, Detect Circularity Part 0 is performed. This algorithm results in separating paths, in which node X_i appears at least in two positions (*i.e.* circular nodes are found) in the strands and pouring them into tube Ticir. Next, for instance, if we assume that three circular nodes X_2, X_3, and X_4 are found, three pairs (X_2, X_3), (X_2, X_4), and (X_3, X_4) should be analyzed in subsequent parts of our algorithms. Using k to represent the number of pairs from circular nodes other parts ofour algorithm is represented in **Table 1**.

Detect Circularity Part 1: At this part of our algorithm for each pair from circular nodes, the algorithm is performed in parallel as follows. Three extra tubes (T_{ij}^a, T_{ij}^b, T_{ij}^c) are necessary for each pair (X_i, X_j). Initially, k copies of T_i cir is created as tubes T_{ij} in parallel. Lines 4 to 9 are carried out until there is no strand in tube T_{ij}. At line 6, strands from T_{ij} having node X_i at position q∗24 are extracted and poured into tube T_{ij}^a. At line 7, strands from T_{ij}^b that all of them has occurrence of X_i are extracted and poured into tube T_{ij}^c if they have X_j located after the first position of X_i. Strands in T_{ij}^c represent paths having X_j located between first and last position of X_i (strands in T_{ij} include at least two occurrence of node X_i). Within the q^{th} iteration of the algorithm, line 5 checks whether T_{ij}^c contains any strands. If it is so, the algorithm makes sure that there exist paths that X_j is located between first and last position of node X_i and part one of the algorithm finishes; otherwise, it will continue until tube T_{ij} contains no strands. If tube T_{ij}^c contains no strands. After performing this part, it means that there is no rules or chains of rules between these two circular nodes acting reverse (which cause circularity still remains in the rule base after performing Detect Circularity Part 0). Thus by removing strands in T_i^{cir} and T_j^{cir} from tube T, the remainder paths contain no circularity dependent rules caused by these two nodes. And performing Detect Circularity part 0 is enough in order to remove circularity. The next part of the algorithm is performed. If tube T_{ij}^c contains any strand. The second part of the algorithm is performed in parallel as follows.

Detect Circularity Part 2: At this part of our algorithm for each pair from circular nodes, the algorithm is performed in parallel as follows. Three extra tubes (T_{ji}^a, T_{ji}^b, T_{ji}^c) are necessary for each pair (X_i, X_j). Initially, k copies of T_j^{cir} is created as tubes T_{ji} in parallel. In this part, between lines 4 to 9, for all pairs of circular nodes in parallel, if there exist a strand in T_{ji}, which has sub-strand representing X_i located between two position of sub-strand representing X_j, is poured into tube T_{ji}^c. There is no rules or chains of rules acting reverse between circular nodes X_i and X_j If there is no strand in T_{ji}^c. In this situation, by removing strands in tubes T_i^{cir} and $T_j^{cirfrom}$ T, we will be sure that there is no circularity error considering nodes X_i and X_j; otherwise, we put into practice the third part of our algorithm for each pair from circular nodes, which both previous parts has been fulfilled for them as follows.

Table 1. Detect circularity algorithm.

{k: number of pairs from circular nodes, in which X_i is the first node in them}	{k: number of pairs from circular nodes defined in previous part, in which X_j is the second node in them}	{k: number of pairs from circular nodes obtained from previous parts}
Detect Circularity Part 1	**Detect Circularity Part 2**	**Detect Circularity Part 3**
1. Input (T_i^{cir}, (X_i,X_j)) for each pair from circular nodes. 2. q=1 3. Copy (T_i^{cir}…,T_{ij}.., T_{ik}), 4. Do while detect (T_{ij})=(Y) 5. If detect (T_{ij}^c)=(N) In parallel : 6. Separate (T_{ij}, X_i, q*24,T_{ij}^a,T_{ij}) 7. Separate (T_{ij}^b, X_j, q*24, T_{ij}^c,T_{ij}^b) 8. Union (T_{ij}^b,T_{ij}^a) 9. q=q+1 10. Else go to part 2 11. End do 12. Remove (.., T_{ij},..T_{ik})	1. Input (T_j^{cir}, (X_i, X_j)) for each pair from circular nodes. 2. q=1 3. Copy (T_j^{cir}, .., T_{ji},.., T_{jk}), 4. Do while detect(T_{ji})=(Y) 5. If detect(T_{ji}^c)=(N){ In parallel : 6. Separate(T_{ji}, X_j, q*24,T_{ji}^a,T_{ji}) 7. Separate(T_{ji}^b, X_i, q*24,T_{ji}^c,T_{ji}^b) 8. Union(T_{ji}^b,T_{ji}^a) 9. q=q+1 } 10. Else go to part 3 11. End do 12. Remove (.., T_{ji},..T_{jk})	1 Input (T,(X_i,X_j)) for each pair of circular node obtained from Part 1 and Part 2. 2. Copy (T,..., T_i,.., T_k) 3. q=1 4. Do while Detect (T_i)=(Y) 5. Separate (T_i, GGGGGGGGGG, In parallel: q*24, T_i^s, T_i) 6. Separate (T_i, X_i, (q*24),T_i^{tmp},T_i) 7. Separate (T_{i1}, X_j, (q*24), T_{i2},T_{i1}) 8. Union (T_{i1}, T_i^{tmp}) 9. q=q+1 10. End Do 11. Remove (..,T_i,..T_k) 12. Union (T^2,.., T_{i2},.., T_{k2})

Detect Circularity Part 3: At this part of the algorithm four extra tubes are necessary for each pair (X_i, X_j) from circular nodes. Initially, k copies of T are generated as tubes T_i. At the final part of our Detect Circularity algorithm for each pair from circular nodes (X_i, X_j), X_i or X_j is selected (here X_i is chosen). Then between lines 4 to 9, all paths in which node X_j is located after $X_i$are extracted from T_i and poured into T_{i2} in parallel. At the end of this part of our algorithm, tubes T_{i2} are merged into tube T^2. We then extract strands in T^2 from tube T. Thus, all complete paths from T, in which node X_j is located after node X_i will be removed from T. Thus, we make sure that after performing this part, there are no rules or chains of rules from X_i leading to X_j in tube T. Consequently one of rules or one of chains of rules causing circularity between X_i and X_j are removed. In the end, the resulting rule base will be free of any form of circularly dependent rules.

In order to demonstrate effectiveness of the algorithm, we perform it for rule base shown in **Figure 5**. As stated above, in this rule base, rules {R_5, R_6, R_7} cause circularity between nodes {X_2, X_3, X_4}. After performing Detect Circularity Part 0, each tube T_i^{cir} has strands shown below.

$$T_2^{cir} = \{R_1R_4R_5R_6R_7\} \qquad T_3^{cir} = \{R_3R_5R_6R_4R_9\} \qquad T_4^{cir} = \{R_2R_6R_4R_5R_8\}$$

These strands are removed from T. In order to clarify the procedure of our algorithm, we perform it for each pair of circular nodes {(X_2, X_3), (X_2, X_4), (X_3, X_4)} successively. In Detect Circularity Part 1, tubes T_{23}, T_{24}, and T_{34} are created for these pairs of circular nodes in parallel at line 3. Tubes T_{32}, T_{42}, and T_{43} are created for these pairs of circular nodes in parallel at Detect Circularity Part 2, line3. First consider nodes (X_2, X_3). By performing Detect Circularity Part 1, In $T_2^{cir}(T_{23})$, we find X_3located between two position of X_2. Thus, Detect circularity part 2 is performed and X_2 is found to be located between two positions of X_3 in $T_3^{cir}(T_{32})$. Thus, there exists rules (chains of rules) acting reverse between these nodes (*i.e.* {R_5, (R_6, R_7)}). At Detect circularity part 3, we choose X_2 and extract all strands, in which X_2 is located before X_3 from T_1 and pour them into T_{12}. We remove these strands from T. Paths 2, 3, and 7 are removed from T and the remainder paths are as follows.

1: R_2R_8: $X_1 \rightarrow X_2 \rightarrow X_5$
5: R_3R_9: $X_1 \rightarrow X_4 \rightarrow X_5$
6: $R_3R_7R_8$: $X_1 \rightarrow X_4 \rightarrow X_2 \rightarrow X_5$
9: R_4R_{10}: $X_1 \rightarrow X_3 \rightarrow X_5$
10: $R_4R_6R_9$: $X_1 \rightarrow X_3 \rightarrow X_4 \rightarrow X_5$
11: $R_4R_6R_7R_8$: $X_1 \rightarrow X_3 \rightarrow X_4 \rightarrow X_2 \rightarrow X_5$

Now, we perform the algorithm for nodes (X_2, X_4). We perform Detect circularity part 1 for node X_2 and in tube $T_2^{cir}(T_{24})$, we find X_4located between two position of node X_2 besides finding (in tube $T_4^{cir}(T_{42})$) X_2 located between two position of node X_4, in Detect circularity part 2. In Detect circularity part 3, we choose node X_2 and extract from tube T_2 all strands having X_2 located before X_4 in their sequence. These strands (if there exist any) should be removed from T. In the end, we perform the algorithm for nodes (X_3, X_4). Since in Detect Circularity Part 1, we find paths in which (in tube $T_3^{cir}(T_{34})$), X_4 is located between two position of node X_3 in addition to finding paths, in which X_3 is located between two position of X_4 in tube $T_4^{cir}(T_{43})$ in Detect Circularity Part 2, we choose one of these nodes (here we choose X_4) and remove all strands, in which X_4 is located before X_3. Consequently the remainder paths and the directed graph made by them are shown in **Figure 6**.

As it is obvious, rule R_4 is removed and circularity error is eliminated from the rule base. Consequently, there is no circle between rules in the resultant rule base. It should be noted that Detect Circularity Part 3 (in lines 5 and 6), depending on the selection of the node that is located before the other in strands, extracts strands in which selected circular node is located before the other circular node (for instance, X_2 is located before X_3 in our example). Therefore, at least one of rule chains (rules) causing circularity between circular nodes is removed and

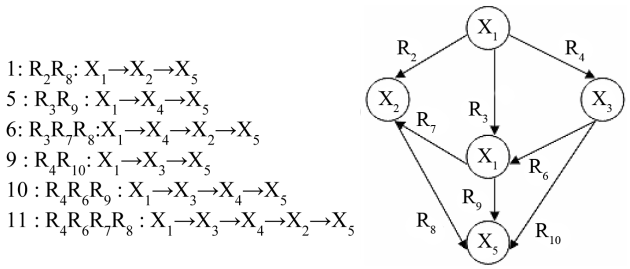


Figure 6. Resultant rule base after removing circularity.

at the most, all the rule chains (rules) causing circularity between circular nodes are removed (in our example all rules {R_4, R_5, R_6} at the most). This removal is dependent on the node selection in Detect Circularity Part 3. however, all pairs from circular nodes, which fulfill Detect Circularity Part 1 and Part 2, has distinct rules or chains of rules from starting nodes leading to each one of them. Thus, our algorithm does not cause incompleteness in the rule base in all cases.

2.6.3. Inconsistency Detection of Rule Bases

Conflicts are known conditions in the system. Thus, we define conflicting nodes as pair (X_i, X_j). If there exists a physical state for which the rules resulting in conflicting nodes can be fired simultaneously, one can say that the rules are physically (practically) conflicting [4]. Our Detect Conflict algorithm is performed for each pairs of conflicting nodes in parallel as follows (assuming k is the number of conflicting pair nodes).

Detect Inconsistency

1. Input (T)
2. Copy (T,… T_z,.., T_k), {k number of pairs of conflicting nodes (X_i, X_j)

In parallel :

3. Extract (T_z, X_i, T_z^+,T_z^-)
4. Extract (T_z^+, X_j,T_{z2},T_{z1})
5. Extract (T_z^-, X_j, T_{z3}, T_{z4})
6. Union (T_r, …, T_{z2},…, T_{k2})

For each pair of conflicting nodes (X_i, X_j) in parallel, strands containing node X_i are extracted from T_z and poured into tube T_z^+; otherwise, poured into tube T_z^- (line 3) .Strands containing X_j are extracted from T_z^+ and poured into Tube T_{z2}; otherwise, into tube T_{z1} (line 4). Strands containing X_j are extracted from T_z^- and poured into T_{z3}; otherwise, into T_{z4} (line 5). Strands in T_{z1} and T_{z3} contain X_i and X_j within their sequences respectively. Strands in tube T_{z2} contain conflicting nodes X_i and X_j in their chain and are invalid. These tubes are merged into tube T_r to be discarded. According to the definition of inconsistency [4], two rules resulting in inconsistent nodes (X_i, X_j) (and consequently their corresponding rule paths in tubes T_{z1}, T_{z3}) are inconsistent, if there exists a state, such that simultaneously both rules can be fired. Although, this is a potential inconsistency, such a possibility exists. These kinds of rules should be further analyzed by domain experts. If there exist a physical state for which the rules can be fired simultaneously, they should be analyzed, modified, or one of them be eliminated. Modification is carried out to arrive at pre-conditions (antecedents) for these rules that cannot be fired at the same time. It is the problem of so called conflict resolution mechanism to select a single rule to be fired [4]. Conflict situations can be solved with appropriate inference control mechanism [4]. If one wants to keep both of these rules, this can be done by controlling the facts and inference control and avoid generating inconsistency by selection of one of the rules (never fire the second rule if the other was fired (e.g. by priority mechanism) [4]. In the event that if, it is decided to eliminate one of the rules resulting in inconsistent nodes (X_i, X_j), this can be done by removing strands in one of these tubes (strands in tube T_{z1} or T_{z3}) from tube T (based on domain experts decision). In this way, inconsistency can be eliminated and the resultant rule base will be free of inconsistency error.

2.6.4. Redundancy Detection of Rule Bases

According to definition of redundancy, a rule is redundant with respect to the conclusion, in the even that if two rules have identical conditions and conclusions identical rules) or two rules have identical conclusions while the condition for one rule is either a generalization or special case of the condition for the other one [3] [4]. The rule, which has more general condition subsumes (is stronger than) the other rule. Suppose that we have more than one starting node in rule base and starting nodes are logically independent and none of them implies the other. For instance, consider the simple rule base and directed graph established for it in **Figure 7**. Assume nodes X_1, X_2, X_3, and X_4 are starting nodes and X_8 is the goal node.

Complete paths from starting nodes leading to goal node are as follows.

According to the definition of redundancy, paths 1, 2, 4 are not redundant from paths 3. Thus, in order to maintain the completeness of rule base, both rule paths 1 and 3 (rules R_5 and R_8) are system required and should remain in rule base (since these rules are not subsumed by any other rule in this rule base). However, algorithm proposed in [1] considers these rule paths redundant from each other. In the subsequent section we propose an

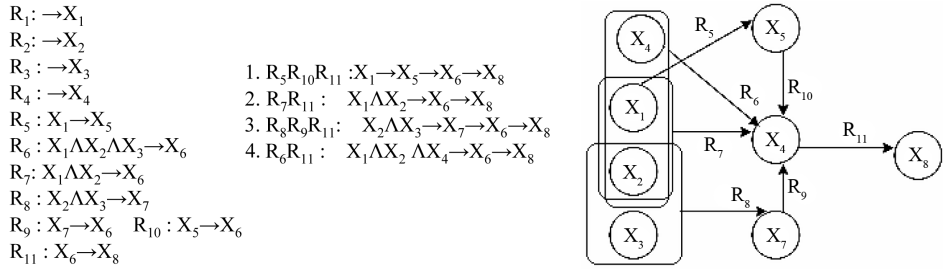


Figure 7. Resultant rule base after removing circularity.

algorithm to detect redundancy in which we have taken into account starting nodes and assumed independency of starting nodes in detecting redundancy error. In our algorithm we will find redundant rules in forms of identical and subsumed rules as follows. The detect redundancy algorithm is represented in **Table 2**.

Initially, z copies of tube T are generated as tubes T_i. At line 2, in parallel for each redundant node, we extract all strands containing redundant node X_i. Next strands containing "$\rightarrow X_i$" are extracted from T_i^+ and poured into tube T_i^{Red}; otherwise, into tube T_i^{Req} at line 3. It should be noted that if a rule path contains the redundant node X_i, but it lacks a node that directly infers X_i, then X_i must be part of a compound AN in the rule path. Thus, this rule path needs to exist when considering redundant node X_i and is poured into tube T_i^{Req} (line 3). Next, we categorize strands in tube T_i^{Red} in terms of length of the antecedent of the first rules in strands. To do so, many copies of strands representing complements of strands in tube $T_{\Lambda sk}$ are poured into tube T_i^{Red}. First, for correct extraction of strands, this line is carried out for tube which contains paths with longest AN of first rules and is repeated until the last tube which contains paths with shortest AN of first rules. Thus, any strand in tube T_i^{Red} that anneals to the above strands are extracted and poured into tube T_{ik}^{Red} (tube $T_{\Lambda sk}$ is comprised of strands representing k starting nodes in conjunction form). Thus, the AN of first rules of strands in tube T_{ik}^{Red} is comprised of k starting nodes, if there exist any strand of this form.

Strands in each tube T_{ik}^{Red}, which the AN of first rules of them, are the same (or permutation of one another) are redundant from each other (antecedent of the first rule in all strands in tube T_{ik}^{Red} have k starting nodes). Other strands in tubes T_{ik}^{Red}, represent rule paths with distinct starting nodes, or strands in which at least one of starting nodes of the first rules are distinct from those of other rule paths. These strands are not redundant from each other.

In order to determine subsumed rules, we compare strands in each tube T_{ik}^{Red} with strands in all other tubes $\{T_{ij}^{Red},\ldots\}$ for all $j > k$, generated at Detect Redundancy Part 1. Our Detect Redundancy Algorithm Part 2 is described below. For each tube T_{ik}^{Red}, this algorithm is performed in parallel as follows. Each strand in tube $T_{\Lambda sk}$ is represented by k_i. Thus, k_z denotes strand number z of tube $T_{\Lambda sk}$ (explained in Section 3.4). At line 3, strands containing k_z are extracted from T_{ik} Red and poured into tube T_{ik}^+. At line 4, If there exists any strand in tube T_{ik}^+, the first rule of this strand subsumes all rule paths in tubes T_{ij}^{Red} ($j > k$), which contain kz. Thus, all rule paths containing kz are extracted from all T_{ij}^{Red} ($j>k$) in parallel and poured into tube $T_{ij}R^+$ (line 5). Consequently all strands in tubes $T_{ij}R^+$, are subsumed rule paths. In order to show the effectiveness of our algorithm, we perform it for the rule base depicted in **Figure 7**. Node X_6 has more than one rule leading to it thus it is candidate for redundancy error. Paths {1, 2, 3, 4} contain sub-strand "$\rightarrow X_6$" and are extracted from T6 and poured into T_6^{Red} at Detect Redundancy Part 1. Next, strands in tube T_6^{Red}, which contain, strands in tube $T_{\Lambda s3}$ (*i.e.* $X_1 \Lambda X_2 \Lambda X_4$) are extracted from T_6^{Red} and poured into tube T_{63}^{Red}. At next iteration of while-loop, strands in tube T_6^{Red}, which contain sub-strands in tube $T_{\Lambda s2}$ (*i.e.* $X_1 \Lambda X_2$ and $X_2 \Lambda X_3$) are extracted from T_6^{Red} and poured into tube T_{62}^{Red}. Finally, strands containing the sub-strand in tube $T_{\Lambda s1}$ (*i.e.* X_1) are extracted and poured into tube T_{61}^{Red}. At Detect Redundancy Part 2, lines 3 to 5 is performed in parallel for all tubes T_{61}^{Red}, T_{62}^{Red}, T_{63}^{Red}. We describe the process for tube T_{61}^{Red}. At line 3, strands in tube T_{61}^{Red}, which contains sub-strand in tube $T_{\Lambda s1}$ (X_1) are extracted from T_{61}^{Red} and poured into tube T_{61}^+. Since tube T_{61}^+ is not empty, line 5 of the algorithm is performed and strands containing Sub-strand X1 are extracted from T_{62}^{Red} and T_{63}^{Red} and poured into tubes $T_{62}R^+$, $T_{63}R^+$ respectively. Finally, strands in tubes $T_{62}R^+$ and $T_{63}R^+$ are redundant (subsumed) rule paths. It should be noted that the algorithm is performed for all tubes T_{ik}^{Red} (in our example, T_{61}^{Red}, T_{62}^{Red}, and T_{63}^{Red}). Thus, between lines 2 to 5, the algorithm will find other redundant rule paths by checking tubes T_{62}^{Red} and T_{63}^{Red}, if there exists any. The process of our algorithm for the example explained above is depicted in **Figure 8(a)**.

Table 2. Detect redundancy algorithm.

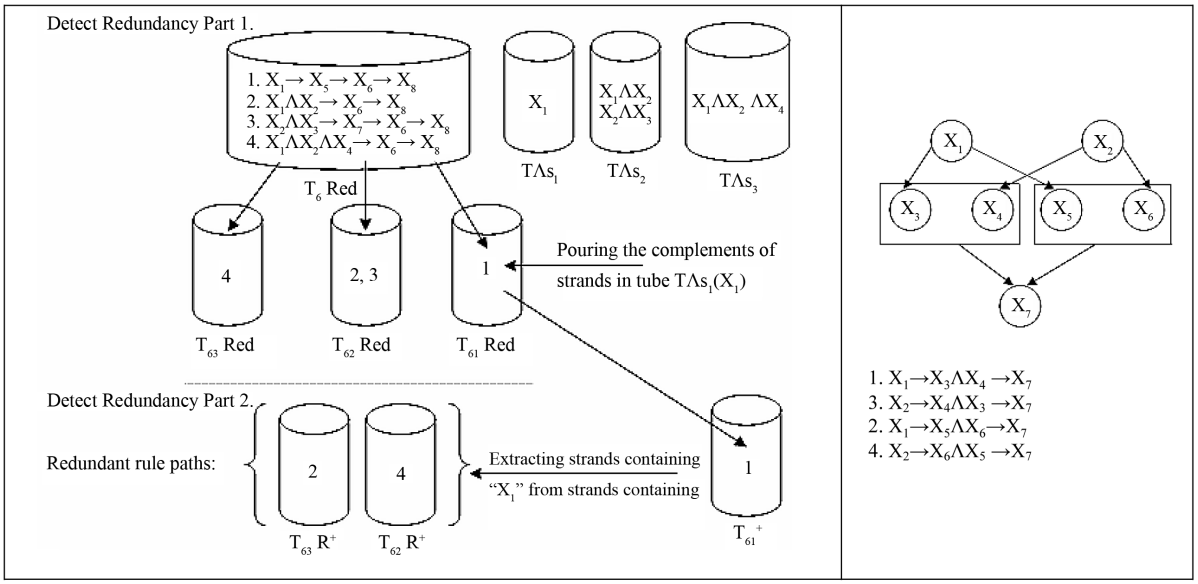
{k = number of last tube from tubes $T_{\Lambda sk}$, arranged in ascending order of k}

Detect Redundancy Algorithm Part 1

1. Copy (T_1,..T_i,..T_z) {z= number of redundant nodes in T_{r3}}
2. Extract (T_i, X_i, T_i^+, T_i^-)
3. Extract (T_i^+,→ X_i, T_i^{Red}, T_i^{Req})
4. In parallelDo while k> 0
5. Extract (T_i^{Red}, $T_{\Lambda sk}$, T_{ik}^{Red}, T_i^{Red})
6. k=k-1
7. End Do

Detect Redundancy Algorithm Part 2

1. Input (T_{ik}^{Red}, $T_{\Lambda sk}$)
2. For z=1 to number of strands k_z in tube $T_{\Lambda sk}$
3. In parallel: Extract (T_{ik}^{Red}, K_z, T_{ik}^+,T_{ik}^{Red})
4. If (Detect (T_{ik}^+) = y)
5. for j>k : Extract (T_{ij}^{Red}, k_z, $T_{ij}R^+$,T_{ij}^{Red})
6. End



(a) (b)

Figure 8. (a) Redundancy detection; (b) Example of redundant rule base.

It should be noted that, rules in the middle of the rule paths with atomic or compound AN does not influence the redundancy detection procedure even if there exist nodes in the middle of the paths that are inferred from distinct starting nodes. To make these statements more clear, take the rule base depicted in **Figure 8(b)** as an example. Assume that X_1 and X_2 are starting nodes and $X_7$is the goal node. The complete rule paths for this rule base is shown in **Figure 8(b)**.

By performing detect redundancy algorithms, rule paths (1, 2) and (3, 4) are redundant from each other. By removing one of the redundant rule paths, incompleteness will not arise in the rule base. The reason is that, a system required rule that is eliminated by removing one of the paths, remains in other rule paths. For such a special case depicted in **Figure 8(b)**, we should be careful not to eliminate redundant rule paths in a way that a system required rule be removed from the rule base, as a result of elimination of redundant rule paths considering different starting nodes (for example, rule $X_3 \Lambda X_4 \rightarrow X_7$ and $X_4 \Lambda X_3 \rightarrow X_7$ will be removed by elimination of rule paths 1 and 3).

3. Conclusions

Different techniques have been developed in order to represent rule-based systems and detect structural errors in them. Nazareth proposed an approach based on Petri nets to verify rule-based systems [13]. Zhang and Nguyen proposed a tool based on Pr/T net to automatically detect potential errors in a rule-based system [23]. Agarwal and Tanniru utilized incidence matrix of Petri nets for detecting structural errors in rule base [14]. An approach based on hyper-graph to verify rule-based systems is proposed by Ramaswamy *et al*. [10], which utilizes di-

rected hyper-graphs to model rule-based systems' graph and transform the hyper-graph into adjacency matrix. He *et al*. [6] utilized a special class of low level Petri nets (ω-nets) in order to detect the structural errors in rule based systems. In their approach, transitions were used to represent rules, and places of Petri-nets represent conditions and conclusions. Thus, rules that their firing does not result in marking of w-net are considered either redundant or circular rules. The algorithms presented in [6] do not distinguish between redundancy and circularity problem. Algorithms based on DNA computing are proposed in [1] in order to detect the four structural errors in rule-base systems. That paper presents algorithms, which are able to detect structural errors just for special cases with one starting node and one goal node. For rule base, which contains more than one starting node and goal node, structural errors are not removed correctly by utilizing these algorithm. By virtue of the special strand design that we used to encode starting nodes and goal nodes, all complete paths can be extracted from initial solution by performing our detect incompleteness algorithm just one time.

Thus, in case of multiple starting nodes and goal nodes, there is no need to repeat the algorithm. Our algorithm efficiently removes all circularity errors in chains of rules in any form that they may occur. In our approach for each pair of inconsistent nodes (X_i, X_j), rule paths which lead to these nodes are extracted from T and categorized in distinct tubes in parallel. Then, these rule paths and rules resulting in inconsistent nodes should be further analyzed, modified or eliminated based on experts domain decision. If there exists a state for which these rules resulting in inconsistent nodes (X_i, X_j) can be fired simultaneously, and it is decided that one of the rules resulting in these nodes should be removed, this can be done by removing strands in one of the tubes containing X_i or X_j based on experts domain decision. The Detect Redundancy algorithm considers starting nodes, which are logically independent and there is no implied relationship between them. Hence, two rules with the same conclusions, which are inferred under different conditions (different starting nodes) are not considered redundant from each other. And Detect Redundancy algorithm is able to detect subsumed rule paths.

Efforts utilizing traditional measures of complexity such as time and space have been made to characterize DNA computation. Most existing models determine the time complexity of DNA-based algorithms by counting the number of biological steps it take to solve the given problem. We use the strong model of DNA computation for parallel filtering models. This model considers that a basic operator actually needs a time dependent on the problem size rather than taking constant time to be carried out [16]. The operation time of some of the operators utilized in this paper is presented in [16]. For instance, Union (T, T_1, …, T_n) and Copy (T, T_1, …, T_n) take O(n) time rather than taking constant time , which n is the problem size.

Our algorithm comprises eight parts: Detect Completeness, Detect Circularity Part 0 to Part 3, and Detect Conflict, Detect Redundancy Part 1, and Detect Redundancy Part 2. We assume that the initial library (solution) is already constructed. Issues about constructing the initial library can be found in [16]. Operations of our algorithm are as follows. The Detect Completeness algorithm includes 2 Extract and one Remove operations and the time it takes is from the order of O(n). Detect Circularity Part 0 includes one Copy, (q − 1) Detect, (3∗(q − 1)) Separate, (q − 1) Union, and one Remove operations and takes O(n∗q) time. Detect Circularity Part 1, consists of one parallel Copy, (2∗ (q − 1)) parallel Detect, (2∗ (q − 1)) parallel Separate, (q − 1) parallel Union, and one Remove and takes O(n∗q) time. By the same token, number of operations included in Detect Circularity Part 2, is exactly the same as Detect Circularity Part 1 and takes O(n∗q) time. Detect Circularity Part 3, consists of one Copy, (q − 1) parallel Detect, (3∗(q − 1)) parallel Separate, (q − 1) parallel Union, one Remove, and one Union operations and takes O(n∗q) time. Detect Conflict algorithm consists of one parallel Copy, 3 parallel extract, one parallel Union and takes O(n) time. Detect Redundancy Part 1 algorithm consists of one Copy, (K + 2) parallel Extract, and takes O(K∗n) time, in which "K" is the number of tubes $T_{\Lambda sk}$. Detect Redundancy Part 2 algorithm consists of (2∗z) parallel Extract and (z) parallel detect operations and takes O(z∗n) time, in which "z" denotes the maximum number of distinct sub-strands in tubes $T_{\Lambda sk}$. In the end, one Read operation is performed which takes O(1) time.

Using the strong parallel model of DNA computation, according to the above complexity analysis, the biological operations of our algorithm in the worst case is O (20q + K + 3z + 3), in "q" is the number of rules in the longest inference chain, "K" is the number of tubes {$T_{\Lambda s1}$, …, $T_{\Lambda sk}$}, and "z" denotes the maximum number of distinct sub-strands in tubes $T_{\Lambda sk}$. If we assume that just Detect circularity Part 0 and Part 1 are performed, then the complexity of our algorithm would be O(10q + K + 3z − 1). The time complexity of our algorithm in all cases is O(n∗(Max{q, K, z})). Applicability of utilizing DNA computing to verification of rule-based systems first shown in [1]. But proposed algorithms were not general and there are lots of cases, in which these errors cannot be removed correctly by utilizing their algorithms. In this paper, the deficiencies of the algorithms pre-

sented in [1] are outlined. We have proposed algorithms in which these deficiencies are eliminated. The proposed algorithms are able to detect the four structural errors in rule base in any forms that they may occur. Our algorithms utilize an entirely linear increase of computation and for a rule base with n nodes, the time complexity of our algorithm is O(n∗(Max{q, K, z})), almost the same as the time complexity that has been achieved in [1]. The number of biological operations used in our algorithm at the worst case of complexity, for which all parts of the Detect Circularity algorithm is performed is O(22q + K + 3z + 3). Our algorithm utilizes some more operations than the algorithm proposed in [1]. However, this small number of added operations results in efficient performance of our algorithms for different cases and consequently, supplements DNA computing approach to verify Rule-Based Systems. In future, we plan to investigate the application of sparse Bayesian models in classification of errors in rule-bases systems [24]-[26]. Additionally we plan to investigate application of system dynamics modeling in implementation and verification of rule based systems [27].

References

[1] Yeh, C.-W. and Chu, C.-P. (2008) Molecular Verification of Rule-Based Systems Based on DNA Computation. *IEEE Transactions on Knowledge and Data Engineering*, **20**, 965-975. http://dx.doi.org/10.1109/TKDE.2007.190743

[2] Jeffrey, J., Lobo, J. and Murata, T. (1996) A High-Level Petri Net for Goal-Directed Semantics of Horn Clause Logic. *IEEE Transactions on Knowledge and Data Engineering*, **8**, 241-259. http://dx.doi.org/10.1109/69.494164

[3] Yang, S.J.H., Tsai, J.P. and Chen, C.C. (2003) Fuzzy Rule Base Systems Verification Using High-Level Petri Nets. *IEEE Transactions on Knowledge and Data Engineering*, **15**, 457-473. http://dx.doi.org/10.1109/TKDE.2003.1185845

[4] Ligeza, A. (2006) Logical Foundations for Rule-Based Systems. Springer, New York, 189-211. http://dx.doi.org/10.1007/3-540-32446-1_12

[5] Tan, Z.-H. (2006) Fuzzy Metagraph and Its Combination with the Indexing Approach in Rule-Base Systems. *IEEE Transactions on Knowledge and Data Engineering*, **18**, 829-841. http://dx.doi.org/10.1109/TKDE.2006.96

[6] He, X., Chu, W.C., Yang, H. and Yang, S.J.H. (1999) A New Approach to Verify Rule-Based Systems Using Petri Nets. *The* 23*rd Annual International Computer Software and Applications Conference*, Phoenix, 27-29 October 1999, 462-467.

[7] Nguyen, T.A. (1987) Verifying Consistency of Production Systems. 3*rd IEEE Conference on Artificial Intelligence and Applications*, Orlando, 4-8.

[8] Nguyen, T.A., Perkins, W.A., Laffey, T.J. and Pecora, D. (1985) Checking Expert System Knowledge Bases for Consistency and Completeness. *Ninth International Joint Conferences on Artificial Intelligence*, Los Angeles, August 1985, 375-378.

[9] Cragun, B.J. and Steudel, H.J. (1987) A Decision-Table-Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems. *International Journal of Man-Machine Studies*, **26**, 633-648. http://dx.doi.org/10.1016/S0020-7373(87)80076-7

[10] Ramaswamy, M., Sarkar, S. and Chen, Y.S. (1997) Using Directed Hypergraphs to Verify Rule-Based Expert Systems. *IEEE Transactions on Knowledge and Data Engineering*, **9**, 221-237. http://dx.doi.org/10.1109/69.591448

[11] Nazareth, D.L. and Kennedy, M.H. (1991) Verification of Rule-Based Knowledge Using Directed Graphs. *Knowledge Acquisition*, **3**, 339-360. http://dx.doi.org/10.1016/S1042-8143(05)80024-X

[12] Valiente, G. (1993) Verification of Knowledge Based Redundancy and Subsumption Using Graph Transformations. *International Journal of Expert Systems*, **6**, 341-355.

[13] Nazareth, D.L. (1993) Investigating the Applicability of Petri Nets for Rule-Based System Verification. *IEEE Transactions on Knowledge and Data Engineering*, **4**, 402-415. http://dx.doi.org/10.1109/69.224193

[14] Agarwal, R. and Tanniru, M. (1992) A Petri Net Based Approach for Verifying the Integrity of Production Systems. *International Journal of Man-Machine Studies*, **36**, 447-468. http://dx.doi.org/10.1016/0020-7373(92)90043-K

[15] Wu, C.H. and Lee, S.J. (1997) Knowledge Verification with an Enhanced High-Level Petri-Net Model. *IEEE Expert*, **12**, 73-80.

[16] Amos, M. (2004) Theoretical and Experimental DNA Computation. Springer, New York, 46-77.

[17] Feynman, R.P. and Gilbert, D. (1960) There's Plenty of Room at the Bottom. *Engineering and Science Magazine*, **23**, 22-36.

[18] Adleman, L.M. (1994) Molecular Computation of Solutions to Combinatorial Problems. *Science*, **266**, 1021-1024. http://dx.doi.org/10.1126/science.7973651

[19] Breslauer, K.J., Frank, R., Blocker, H. and Marky, L.A. (1986) Predicting DNA Duplex Stability from the Base Sequence. *Proceedings of the National Academy of Sciences*, **83**, 3746-3750. http://dx.doi.org/10.1073/pnas.83.11.3746

[20] Braich, R.S., Johnson, C., Rothemund, P.W.K., Hwang, D., Chelyapov, N. and Adleman, L.M. (2001) Solution of a Satisfiability Problem on a Gel-Based DNA Computer. *DNA Computing Lecture Notes in Computer Science*, **2054**, 27-42.

[21] Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K. and Adleman, L.M. (1998) A Sticker-Based Model for DNA Computing. *Journal of Computational Biology*, **5**, 615-629. http://dx.doi.org/10.1089/cmb.1998.5.615

[22] Chang, W.-L. and Guo, M. (2004) Molecular Solutions for the Subset-Sum Problem on DNA-Based Supercomputing. *Biosystems*, **73**, 117-130. http://dx.doi.org/10.1016/j.biosystems.2003.11.001

[23] Zhang, D. and Nguyen, D. (1994) PREPARE: A Tool for Knowledge Base Verification. *IEEE Transactions on Knowledge and Data Engineering*, **6**, 983-989.

[24] Madahian, B., Deng, L. and Homayouni, R. (2014) Application of Sparse Bayesian Generalized Linear Model to Gene Expression Data for Classification of Prostate Cancer Subtypes. *Open Journal of Statistics*, **4**, 518-526. http://dx.doi.org/10.4236/ojs.2014.47049

[25] Madahian, B. and Faghihi, U. (2014) A Fully Bayesian Sparse Probit Model for Text Categorization. *Open Journal of Statistics*, **4**, 611-619. http://dx.doi.org/10.4236/ojs.2014.48057

[26] Madahian, B., Deng, L. and Homayouni, R. (2014) Development of Sparse Bayesian Multinomial Generalized Linear Model for Multi-Class Prediction. *BMC Bioinformatics*, **15**, P14. http://dx.doi.org/10.1186/1471-2105-15-S10-P14

[27] Madahian, B., Klesges, R.C., Klesges, L. and Homayouni, R. (2012) System Dynamics Modeling of Childhood Obesity. *BMC Bioinformatics*, **13**, A13. http://dx.doi.org/10.1186/1471-2105-13-S12-A13

Scientific Research Publishing (SCIRP) is one of the largest Open Access journal publishers. It is currently publishing more than 200 open access, online, peer-reviewed journals covering a wide range of academic disciplines. SCIRP serves the worldwide academic communities and contributes to the progress and application of science with its publication.

Other selected journals from SCIRP are listed as below. Submit your manuscript to us via either submit@scirp.org or Online Submission Portal.