An Improvement of Requirement-Based Compliance Checking Algorithm in Service Workflows

Wattana Viriyasitavat
Business Information Technology Division, Department of Statistics, Faculty of Commerce and Accountancy, Chulalongkorn University, Bangkok, Thailand
hardgolf@gmail.com

Andrew Martin
Department of Computer Science, University of Oxford, Oxford, United Kingdom
andrew.martin@cs.ox.ac.uk

Abstract—This paper presents an improvement of requirement-oriented compliance checking algorithm to support trust-based decision making in service workflow environments. The proposed algorithm is based on our previous progressive works on (1) Service Workflow Specification language (SWSpec) serving as a formal and uniformed representation of requirements, and (2) the algorithm based on Constrained Truth Table (CTT), specifically developed for compliance checking for the Composite class of SWSpec. However, CTT algorithm practically suffers from high complexity which is \(O(|S||V|^2)\), where \(|V|\) is the number of services presented in a workflow, and \(|S|\) is the size of a SWSpec formula to be checked. In this paper, we improve algorithm CTT by using Exclusive Disjunctive Normal Form (EDNF) as a new data structure that reduces the time complexity in the average case to \(O(|S||V|^2)\). Finally, the performance comparison between these two approaches is conducted.

Keywords—Service, Workflow, Compliance Checking

I. INTRODUCTION

Service workflows have received much interest in the past decades. Nowadays, they appear in several forms ([1], [2], and [3]). For example, within an organization, services are used as a building block to streamline and automate business processes to improve efficiency and scalability. In decentralized collaborative environments such as Grids [4], Virtual Organizations (VO), and Cloud Computing, services become a fundamental element for collaborations. Despite their wide range of applications, services still suffer from the lack of an agreed and standard in requirement representation.

Formal methods provide rich specification languages ([5], [6], and [7]), to express such requirements, modeling languages to abstract systems to be verified, and algorithms. To achieve automatic reasoning that is needed to facilitate scalability, dynamicity, and security in large-scale open environments, three essential elements are required: (1) a modeling language in which workflows can be logically abstracted to represent structure of services, tasks, and their relationships, (2) a specification language as a formal and uniformed representation of requirements, and (3) compliance checking algorithms to ascertain that the services satisfy such requirements [8]. All of the three elements have been comprehensively addressed in our previous work. The workflow to be verified is modeled by Service Workflow Net (SWN), with the introduction of control connectives for structure formulation; the requirements are formally represented by SWSpec formulas [9]; and the compliance checking algorithm are developed based on CTT [10]. In this paper, we improve the algorithm CTT by using EDNF as a new data structure that reduces the time complexity from \(O(|S||V|^2\)} to \(O(|S||V|^2)\) in the average case.

The rest of this paper is organized as follows. Section II presents our previous work on SWN, SWSpec, and algorithm CTT. Section III explains the process of simplifying SWSpec formulas, which will be used for EDNF compliance checking algorithm. In section IV, algorithm EDNF is presented. Then, we conduct the analysis with comparison of performance between CTT and EDNF algorithms in Section VI. Section VII presents some related works, and the last section concludes and discusses potential future research.

II. BACKGROUND

In this section, the information regarding (1) our workflow modeling, SWN, (2) SWSpec formulas, and (3) CTT algorithm is described shown in Table I, II, and III, respectively. Please refer to [9] and [10] for more information and justifications.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>SWN DEFINITION</th>
</tr>
</thead>
</table>

Def. 1 A Petri Net is a labeled Place/Transition Net, i.e., a 7-tuple \( M = (P, T, R, f, l, o, I) \), where:
1. \( P \) is a set of places (representing services),
2. \( T \) is a set of transitions (representing tasks),
3. \( R \subseteq (P \times T) \cup (T \times P) \) represents directed flows,
4. \( f : (P \rightarrow (\phi \times \{\text{split}, \text{join}\})) \cup (T \rightarrow (\phi \times \{\text{split}, \text{join}\})) \) is a function containing a workflow structure formula (\( \phi \)) with either a split or a join type. A formula contains three types of connectives (AND, OR, and XOR).
5. \( I: P \rightarrow A \cup \{\emptyset\} \) is a labelling function where \( A \) is a set of properties, and \( \emptyset \) denotes a null value. It is used for labelling a service with attributes (properties).

Service-split (or service composition) is defined as a set of possible services that can be activated for a task execution.

Service-join (or service separation) identified a set of services that can be triggered after the execution is done.
TABLE II

<table>
<thead>
<tr>
<th>SWSpec Grammars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition: A SWSpec grammar $G = (V, T, S_0, R)$ consists of a finite set of variables $V$, a finite set of terminal symbols $T$, a start symbol $S_0$, and a set of production rules $R$. A SWSpec formula is a tree, where the root is a composite quantifier operator, the leaves are cloves, and the internal nodes are composite operators. The SWSpec grammar is used to parse SWSpec formulas from the SWN format into the SWSpec format.</td>
</tr>
</tbody>
</table>

III. PREPROCESSING SWSPEC FORMULAS

For simplicity, any SWSpec formula is translated into a simpler form. It can be pre-processed until the property part of a Composite formula includes only $\forall$ and $\exists$ in order. The notions of cloves and clove trees (from [10]) below represent the transformed formula (see Figure 1).

Def. 4 (Clove): Given a Composite formula, a clove is defined as a set of atomic propositions linked by $\land$ or $\lor$, or a single atomic proposition if no such operator is involved.

Def. 5 (Clove Tree): A clove tree is the representation of a Composite formula with the quantifier part as a root, the second level is $\land$ or $\lor$ operator, and the leaves are cloves.

The formula $\forall (q_1 \land (q_2 \lor (q_3 \land q_4)))$ in Figure 1 can be interpreted as follows (see algebraic properties in Table IV).

![Figure 1. The Clove Tree of a Formula $\forall (q_1 \land (q_2 \lor (q_3 \land q_4)))$ and its Transformation with the Notion of Cloves of $\forall q_1 \land (q_2 \lor q_3)$ or $\forall q_1 \land (q_3 \land q_4))$](image_url)
- Initial form
  \[ P_E(q_1 \cap (q_2 \oplus (q_3 \cup q_4)) \]

- Applying Distributive Property
  \[ \equiv P_E((q_1 \cap q_2) \oplus (q_1 \cap (q_3 \cup q_4)) \]

- Applying elimination
  \[ \equiv P_E(q_1 \cap q_2) \mid P_E(q_1 \cap (q_3 \cup q_4)) \]

- Applying Distributive Property
  \[ \equiv P_E(q_1 \cap q_2) \mid P_E((q_1 \cap q_3) \cup (q_1 \cap q_4)) \]

This transformation makes the algorithm simpler because the absence of \( \oplus \) allows us to circumvent the check between OR and \( \cap \) that can be done indirectly by \( \cap \) and \( \cup \).

### Table III
Reduction Rules

Suppose that \( \phi \) and \( \omega \) are two sub-SWSpec formulas

- **Rule 1:** For \( \phi \) AND \( \omega \), \( \phi \) and \( \omega \) must both be true or false.
- **Rule 2:** For \( \phi \) XOR \( \omega \), \( \phi \) and \( \omega \) cannot both be true.
- **Rule 3:** The results of the evaluation cannot be false.

### Algorithm CTT

1. Function CTT (CTT, V)
   // CTT.R = a constrained truth table object
   // CTT.R = a set of rows
   // V = a set of workflow variables

2. Begin
3. for each \( r_1 \in \text{CTT}.R \)
4. for each variable \( v \in V \) in that row
5. \( b \leftarrow \text{true} \)
6. if all \( v \in V \) marked with 1 \( \equiv \) any clove
7. \( r_1 \leftarrow \text{satisfied} \)
8. \( \text{End if} \)
9. \( \text{End if} \)
10. if \( b \) is presented,
11. if some \( v \in V \) marked with 1 \( \equiv \) any clove
12. \( r_1 \leftarrow \text{satisfied} \)
13. \( \text{End if} \)
14. \( \text{End if} \)
15. \( \text{End for} \)
16. \( \text{End for} \)
17. if all \( r_1 \ in \ v \) is satisfied
18. \( \text{Return satisfied} \)
19. else if some \( r_1 \ in \ v \) is satisfied
20. \( \text{Return partially satisfied} \)
21. else
22. \( \text{Return unsatisfied} \)
23. \( \text{End if} \)
24. \( \text{End Function} \)

### Table IV
Performance Comparison

<table>
<thead>
<tr>
<th>Name</th>
<th>Initial Form</th>
<th>Transformed Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributive</td>
<td>( Z_1 \cap (Z_2 \oplus Z_3) )</td>
<td>( (Z_1 \cap Z_2) \oplus (Z_1 \cap Z_3) )</td>
</tr>
<tr>
<td>Property</td>
<td>( Z_1 \cap (Z_2 \cup Z_3) )</td>
<td>( (Z_1 \cap Z_2) \cup (Z_1 \cap Z_3) )</td>
</tr>
<tr>
<td>Elimination</td>
<td>( E(Z_1 \oplus Z_2) )</td>
<td>( (E(Z_1 &amp; E(Z_2)) \mid (E(Z_1 &amp; \neg E(Z_2))) )</td>
</tr>
</tbody>
</table>

IV. EDNF

Normal form is an alternative choice in representing Boolean functions in a more concise. A formula with the same number of variables is much more compact comparing to CTT. For this reason, effective compliance checking can be developed. Terms in EDNF are all variables that are connected by AND connectives, and XOR connectives are used to connect between terms. If all terms are true, the result is satisfied. If some are true, the result is partially satisfied; otherwise, it is unsatisfied.

To circumvent the check between OR and \( \oplus \) as mentioned earlier, any workflow formula is presented by the combination of AND and XOR, while OR can be transformed as follows:

\[ A \lor B = A \times (A \lor B) \times (A \land B) \]

**Def. 6 (EDNF):** An SWN formula is EDNF if it is an exclusive disjunction of terms where each term is a conjunction of literals.

**A. Algorithm EDNFSAT**

Assume that all SWN formulas are presented in the form of EDNF. The complexity of compliance checking depends on (1) the number of the occurrence of workflow variables (services), (2) the number of connectives, and (3) reasoning algorithms. One of the most efficient algorithms employs a binary tree data structure to represent a workflow formula. Leaf nodes are workflow variables while the upper nodes represent workflow connectives. The graphical explanation of the complexity calculation is illustrated in Figure 2. The operations of this algorithm can be understood by the following steps (the pseudo code for EDNFSat is presented in Table V).

1) Each leaf node is marked with satisfied, unsatisfied, or unknown, if it satisfies, does not satisfies, or partly satisfies with any clove in a clove tree. For instance, if a node contains a property \( q_2 \) and there is a clove \( c_1 = (q_1 \cap q_2) \) which is part of a clove tree of a Composite formula \( P_A(q_1, q_2) \), we mark the node with unknown, where the subscripted \( Q = \{q_2, c_1\} \).

2) For each upper AND node traversing up towards the top AND node in an SWN tree formula, if \( E \) is presented in the clove tree,
   a) if at least one lower node is marked with satisfied, we mark the upper AND node with satisfied,
   b) if one node is unknown and another is marked with unsatisfied, we mark the upper AND node with unknown,
   c) if two lower nodes with unknown marking are combined which results in satisfying any clove, we mark the upper AND node with satisfied. If not, it is marked with unknown,
   d) otherwise, we mark the upper AND node with unsatisfied, and repeat until traversing to the top AND node.
   e) Go to step 4.

3) For each upper AND node traversing up towards the top AND node in an SWN tree formula, if \( A \) is presented in the clove tree,
   a) We mark the node with satisfied, if the lower nodes are the combination of (1) both marked with satisfied, or (2) satisfied and unknown.
TABLE V
ALGORITHM EDNFSAT

1: Function EDNFSAT(SWND) // SWND is an EDNF tree representing workflow formulas in the form of clove tree
2: 2: Q = a set of properties of unknown status to satisfy any clove;
3: 3: V = SWND. V // a set of workflow variables;
4: 4: M = all presented connectives;
5: Begin
6: For each v_i ∈ V, 
7: If v_i = any clove 
8: v_i = satisfied;
9: Else if v_i partly complies with any clove 
10: v_i = unknownQ;
11: Else v_i = unsatisfied;
12: End if
13: End for
14: For each m_i ∈ M and m_i = AND //Assume that m_i is chosen in order from low-to-high layer of the SWND
15: H(m_i) is presented 
16: If (m_i, left = satisfied AND m_i, right = satisfied) OR 
17: (m_i, left = unknownQ AND m_i, right = satisfied) OR 
18: (m_i, left = satisfied AND m_i, right = unknownQ)
19: m_i = satisfied;
20: Else if (m_i, left = unknownQ AND m_i, right = unknownQ)
21: m_i = satisfied;
22: Else m_i = unsatisfied;
23: End if
24: End for
25: If H_m is TOP AND NODE 
26: H_m = unknownQ
27: End if
28: End if
29: Else if H_m is TOP AND NODE 
30: H_m = unknownQ
31: End if
32: For each m_i ∈ M and m_i = XOR //Assume m_i is chosen in order from low-to-high layer of the SWND
33: H_m, left = satisfied AND m_i, right = satisfied 
34: m_i = satisfied;
35: Else if (m_i, left = unknownQ AND m_i, right = unsatisfied) OR 
36: (m_i, left = satisfied AND m_i, right = unknownQ)
37: m_i = unsatisfied;
38: End if
39: End if
40: End for
41: End if
42: End for
50: End for
51: End function

b) if both lower nodes with unknownQ status are combined to satisfy a clove, we mark the node with satisfied, if not, it is marked with unknownQ.

c) if an unsatisfied mark is presented, we immediately mark the top AND node with unsatisfied, and go to step 4.

d) Repeat until traversing to the top AND node.

4) If the top AND node is unknownQ, we remark it with unsatisfied.

5) In an upper node representing XOR connective,
   a) if all lower nodes are marked with satisfied, it is also marked with satisfied,
   b) if one of its lower nodes is marked with satisfied and another with unsatisfied or partially satisfied, it will be marked with partially satisfied;
   c) otherwise it is marked with unsatisfied.

6) Repeat step 5 until reaching the root node where we can determine the final result.

Figure 2. Graphical Representation of Efficiency Complexity Calculation

B. Analysis of EDNFSAT

Assume again the checking operation between a clove and a leaf node occupies one time unit, the best efficiency evaluation of this form is O(|C||T||V||M|) where |C|, |T|, |V|, and |M| are the number of cloves, clove trees, workflow variable, and workflow connectives respectively. In a concise form, |C||T| can be reduced to |S|, representing a size of SWSpec tree, such that the time complexity is presented as O(|S||V||M|). In the worst case scenario, the maximum number of occurrence M of workflow variables V can be calculated by the following equation.

\[ N = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} \]

According to Taylor’s approximation,

\[ (1 + x)^a = \binom{a}{0} + \binom{a}{1} x + \binom{a}{2} x^2 + \cdots + \binom{a}{n} x^n \]

If x = 1 we have

\[ \binom{a}{0} + \binom{a}{1} + \cdots + \binom{a}{n} = 2^a \]

such that,

\[ N = \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{n} = 2^n - 1 \]

As a result, the computational complexity of this form is \( O(|S||V|2^{|V|}) \). However, it is important to look at the average occurrence of V that can be computed as the following equation.

\[ \text{average} = E(n \times \binom{n}{1} + (n - 1) \times \binom{n}{2} + \cdots + 0 \times \binom{n}{0}) \]
Therefore, the average time complexity of EDNFSat is $O(|S||V|^{2n})$. The performance comparison between EDNFSAT and CTTSAT is presented in Table VI.

<table>
<thead>
<tr>
<th>Models</th>
<th>Checking Time</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constrained Truth Table</td>
<td>$O(</td>
<td>S</td>
</tr>
<tr>
<td>EDNFSat</td>
<td>$O(</td>
<td>S</td>
</tr>
</tbody>
</table>

V. PERFORMANCE EVALUATION

To confirm the applicability, we have developed a prototype to validate our framework. All functions are written in MATLAB to demonstrate the proof of concept and performance comparison between two approaches. The system runs on a Windows 7, Intel® Core™ i5-2435M CPU @ 2.40 GHz, 4 GB RAM, 64-bit Operating System. We design the experiment to evaluate time performance when 10, 50, and 100 services are involved. The result in Figure 3(a) shows that the performance between algorithm EDNFSAT and CTTSAT in the worst case scenario is similar. However, EDNFSAT runs faster in the average case (see Figure 3(b) and Figure 3(c) and (d) for the comparison in bar graph). This corresponds to the theoretical evaluation in Table VI.

![Figure 3](image)

Figure 3. Performance Evaluation between CTTSAT and EDNFSAT

VI. RELATED WORKS

After Model checking is first introduced [11], it has been extended to cover wider domains beyond the specific systems modeled by Finite State Machine. It has spread across many areas ranging from verification between business processes and contacts [12], policy-based compliance checking for trust [13], logic-based verification [14] to hardware and software component testing at very low level. In our essence, we intend to apply the concept of model checking for compliance checking in the service workflow domain and requirements specification.

VII. CONCLUSION

This paper presents algorithm EDNFSAT for compliance checking between SWSpec formulas and service workflow. It improves the existing algorithm CTTSAT that specifically deals with Composite class of SWSpec. We conduct the experiment to compare between these two approaches. The primary advantage of this algorithm is that in average case, time complexity for checking operation is reduced into polynomial. In practice, the checking process can be locally computed; each involved service is only to verify if its own requirements. Furthermore, since SWSpec formulas are independent from each other, using parallel computing will significantly improve the overall performance. For future work, we plan to develop the tracer to indicate the conflict points, if any, in both SWSpec and SWN, and to provide the counter example of this conflict.

REFERENCES