Master Thesis – Math Formalization

Kefang Ding

9 Nov 2018

Abstract

This article defines the mathematical formalization of algorithm, which is used to incorporate negative information for model repair. The following sections are organized in this way. Section 1 introduces the problems to solve. Section 2 defines the symbols. Section 2 describes the algorithm to use. Section 3 proves the correctness and completeness of this algorithm.

1 Introduction

The inputs for process model enhancement within model repair includes the following data, existing process model, event log and KPIs to evaluate the data in event log.

After applying Dfg-based repair model, a model with good fitness is generated. However, this method can't discovery change and remove the long-term dependency in the model.

Long-term dependency describes the dependency in events, where the execution of one event affects the choice of events in later. It exists in the choices structure of model, like xor structure, loop and or structure. But now we only focus on the long-term dependency in xor and loop structure, due to the complexity of or structure in model.

Here we introduce an additional algorithm to deal with the long term dependency. In the next part, we will focus on the algorithm to discover long-term dependency in model.

The input for the algorithm is:

- Repaired model in process tree
- Event log with positive and negative labels

The output of this algorithm is:

- Repaired model in petri net with long-term dependency

2 Definitions

In this section, the related definitions are listed. Firstly, the definition for process tree is reviewed.

Definition 2.1 (Process Tree). *Let $A \subseteq \mathbb{A}$ be a finite set of activities with $\tau \in \mathbb{A}$, $\bigoplus \subseteq \{\rightarrow, \times, \wedge, \cup^1\}$ be the set of process tree operators.*

- *$Q = a$ is a process tree with $a \in A$, and*
- *$Q = \oplus(Q_1, Q_2, ..Q_n)$ is a process tree with $\oplus \in \bigoplus$, Q_i is a process tree, $i \in 1, 2, .., n, n \in \mathbb{N}$.*

Process tree operators represents different block relation of each subtree. Their semantics are listed below.

Definition 2.2 (Operator Semantics). *The semantics of operators $\bigoplus \subseteq \rightarrow, \times, \wedge, \cup$ are,*

- *if $Q = \rightarrow (Q_1, Q_2, ..Q_n)$, the subtrees have sequential relation and are executed in order of $Q_1, Q_2, ..Q_n$*
- *if $Q = \times(Q_1, Q_2, ..Q_n)$, the subtrees have exclusive relation and $Q_1, Q_2, ..Q_n$ only one subtree of them can be executed.*
- *if $Q = \wedge(Q_1, Q_2, ..Q_n)$, the subtrees have parallel relation and $Q_1, Q_2, ..Q_n$ they can be executed in parallel.*
- *if $Q = \cup(Q_1, Q_2, ..Q_n)$, the subtrees have loop relation and $Q_1, Q_2, ..Q_n with n \geq 2$,$Q_1$ is the do-part and is executed at least once, $Q_2, ..Q_n$ are redo part and have exclusive relation.*

In the following figure, it's a typical process tree. It describes a business model, which includes the sequential, exclusive and parallel relations among the activities. The model is sound.

During model execution, we observe that, in some situations, the execution of events in the exclusive block has influence on the execution of events later, which is called long-term dependency. In the following, the focus is to deal with long-term dependency in the model. For the sake of convenience, exclusive block is abbreviated as xor block; also, the subtree of xor block is defined as xor branch.

Definition 2.3 (xor branch). *$Q = \times(Q_1, Q_2, ..Q_n)$, Q_i is one xor branch with respect to Q. For convenience, we use X to represent one xor branch, and record it $X \in XOR_Q$*

Due to the different structure of xor branch, we drive two properties of xor block, purity and nestedness.

[1] it means the loop operator due to difficulty to print out the real loop symbol

Definition 2.4 (XOR Purity). *A xor block is pure if and only* $\forall X \in XOR_Q, Leaf(X) \rightarrow True$. *Else, the block is unpure.*

Definition 2.5 (XOR Nestedness). *A xor block is nested if and only* $\exists X XOR(X) \wedge Ct(XOR_Q, X)$, *where* $Ancestor(XOR_Q, X)$ *represents* XOR_Q *is an ancestor of* X *in the process tree.*

The long-term dependency is associated with each xor branch in xor block. To define it, the following concepts are in need. The first is the order of xor block in sequential structure.

Definition 2.6 (Order of xor block). xor_A *is before* xor_B, *written in* $xor_A \prec xor_B$, *if and only if one ancestor branch of* xor_A *is before the ancestor of* xor_B *in sequential block.*

If the least common ancestors of xor block are parallel, they don't have any order with them. If they are in loop, then the xor in do part is before the xor in reloop part. With the order definition, we introduce the xor pair.

Definition 2.7 (XOR Pair). xor_A *and* xor_B *is an xor pair, written in* $XOR\_Pair(XOR_A, XOR_B)$, *if* $XOR_A \prec XOR_B$.

Definition 2.8 (Event Frequency). *Event Frequency in an event log* l *is an atom* $F_l(a, freq)$ *where* a *is an event,*$a \in A$ *and freq is the happened frequency in integer for event* a.

In the recursive definition from the above, we can define the xor branch frequency in an event log.

Definition 2.9 (Xor Branch Frequency). *Xor branch frequency in event log* l *is* $F_l(X, freq)$ *where* X *is an xor branch, and freq is the happened frequency in integer for xor branch* X.

Definition 2.10 (Supported Connection of Xor branches). *Given an event log, xor branch* X *and* Y *have supported connection over a threshold* t, $SC_l(X, Y, t)$ *if and only if*

$$forX \in xor_S, Y \in xor_T, \exists freq, F_l(X, freq) \wedge F_l(Y, freq) \wedge freq \geq t.$$

After introduction of supported connection of xor branches, we can define the long-term dependency.

Definition 2.11 (Long-term Dependency in XOR block). xor_S *and* xor_T *have long-term dependency over an threshold* t *w.r.t. an event log,* $LT(xor_S, xor_T, t)$ *if and only if*

- *there are at least two xor blocks in model,* $xor_S \neq xor_T \wedge xor_S \prec xor_B$
- $\exists X \in XOR_S, \exists Y \in XOR_T, \neg SC_l(X, Y, t)$.

In this context, we define the long-term dependency between xor branches.

Definition 2.12 (Long-term Dependency in XOR branches)**.** *Xor branch X and Y have long-term dependency over an threshold t w.r.t. an event log, $LT(X, Y, t)$ if and only if*

$$\exists X \in XOR_S, \exists Y \in XOR_T, LT(XOR_S, XOR_T, t) \wedge SC(X, Y, t) \rightarrow LT(X, Y, t).$$

3 Algorithm

According to the long-term dependency definition, we propose our algorithm to discover long-term dependency. Because the purity, nestedness and its position in process tree, we need to deal with long-term dependency in different situations.However, due to the complexity, the algorithm focuses only on the binary long-term dependency of xor block, which means, we only create xor pair of $XOR_X and XOR_Y$,where

$$\exists! XOR_Z,\ XOR_S \prec XOR_T \rightarrow XOR_S \prec XOR_Z \wedge XOR_Z \prec XOR_Y$$

The general steps of algorithm is in the following.

Algorithm 1: General steps to add long-term dependency

```
Result: Discover Long-term Dependency In Model
create a list including all xor pairs in process tree;
while pair in xor pair list do
    if this pair has no LT dependency then
        remove this pair from xor pair list;
    end
end
transfer process tree into Petri net;
add places in Petri net for every branch pair with long-term
 dependency;
```

We give more details about the each steps in the next parts.

3.1 Create All XOR Pairs

Given one process tree with xor blocks, we create XOR pairs in such situations.

- Sequential XOR Block Without Nested XOR Block
- Sequential XOR Block With Nested XOR Block
- Parallel XOR Block
- Loop XOR Block

3.2 Check if the pair has LT Dependency

The first situation is the pure and impure xor block without nested xor block in sequential order. The algorithm is