Formal Verification Algorithm for Workflow Processes

Anna, Varosyan

Virage Logic
15/1 Khorenatsi str., Yerevan, Armenia
e-mail: anna.varosyan@viragelogic.com

ABSTRACT

Workflow process (WFP) functional/behavioral verification has become an essential step in ensuring quality of designed workflows. It can be done either formally or via process simulation. The most commonly used case is a simulation of the process execution for all possible inputs, but this usually leads to exhaustion. To reduce it a formal verification of workflow processes should be applied whenever possible. A paper presents an algorithm of the WFP formal verification without any restriction on the process graph.

Keywords

Workflow Process, formal verification, cycle transformation, mold, approximation techniques.

1. INTRODUCTION

Business Process Management is a key technology in the overall enterprise strategy of an organization [1]. The workflow processes have to be specified by a language called workflow specification language [1, 2]. It is very important to verify the workflow process correctness prior to the operation [3].

The workflow process correctness can be checked by the process graph structural verification, such as cycle analysis, branching path synchronization, etc [3, 4 and 5].

The functional/behavioral verification of a process can be considered as another case of verification to verify the process semantic correctness. This can be done either formally or via process simulation. The most commonly used case of functional verification is a simulation of the process execution for all possible cases, but this can lead to exhaustion. That is the reason why the formal verification of processes should be applied whenever possible.

The formal verification requires knowledge about the underlying workflow process (internal structure of tasks, data flow, etc.). Two assertions have to be specified for the process. First, the process input assertion, called precondition, has to be satisfied prior to the process execution. Second, the process output assertion, called postcondition, has to be checked at the end of each process execution. The process is considered to be correct if the value of postcondition is “true” for all possible executions.

In general, insolubility (or exhaustion) hampers direct application of formal verification to complex cases. Therefore, the desire to apply a strict criterion must be neglected and replaced with finding effectively verifiable, acceptable conditions for verification. More explicitly, algorithms should be constructed, which are applicable to any object considered for verification. These algorithms must give one of the three answers: correct, incorrect, or unknown under restrictions on execution time. An algorithm is called partial-recognizing if it answers correct (incorrect) only for correct (incorrect) objects. A partial-recognizing algorithm (PRA) can answer "the correctness is unknown" in the case of a correct object. But when it answers "the object is correct," the object is correct for certain.

By adopting the described approach to a workflow process (WFP) [1] formal verification, one way to create a PRA is to approximate the input and output assertions of a WFP by special language constructions called molds (primitives for the description of environment state subsets). The idea was initially realized for verification of acyclic programs processing Boolean data [6, 7].

In this paper a formal verification algorithm for workflow processes is suggested. It is also based on molds (primitives for the description of environment state subsets). The two-step solution comprises of transformation of a given cyclic WFP to an acyclic WFP and further application of the technique similar to the above to the obtained acyclic WFP.

The transformation of a process graph is based on determining intervals, suggested for program data flow analysis by F.E.Allen and J.Cocke [8]. This idea, initially used in optimization of compilers, was used afterward in many other applications [9].

An extension of the IBM’s MQSeries Workflow model [1], which provides necessary data for formal verification algorithm, is presented in the paper.

2. AN ALGORITHM OF FORMAL VERIFICATION

The verification algorithm schema is presented in Figure1. It contains two main blocks – cycle transformation and mold construction. At first the existence of cycles has to be checked. If the process is cyclic, the properties of transformability would be checked. In the case of process transformability the cycle transformation block would be activated. Otherwise, a simulation algorithm can be applied [11, 13]. Mold construction block has to be applied after the process transformation or directly, in the case of cycle absence. Finally, the comparison of current mold and postcondition mold would define if the process is correct, incorrect or its correctness is unknown. In the case of ‘unknown’ the simulation algorithm would be applied [11, 13]

The theoretical foundation of mold construction block is presented in [10], while the cycle transformation block is explained in [12]. This section summarizes the obtained results of articles [10] and [12] in one algorithm which is presented in the paper.

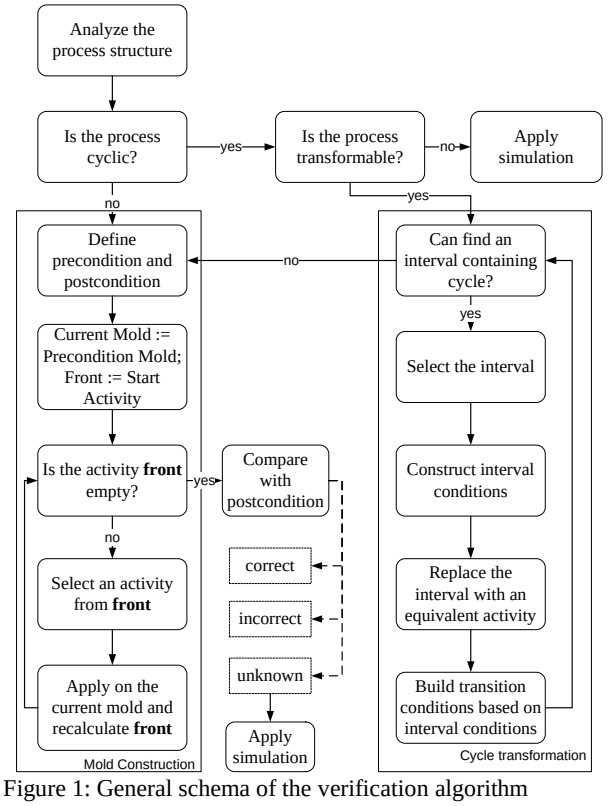

Figure 1: General schema of the verification algorithm

Section 2.1 presents model description that is based on the IBM's MQSeries Workflow model [1]. The cycle transformation basic operations are brought forward for consideration in section 2.2. The next section contains the basic operations of mold construction. Sections 2.4 and 2.5 present the descriptions of transformation conditions and the verification algorithm.

2.1. Model Description

The formal workflow/BPM model underlies a model of IBM's MQSeries Workflow. The formal definition of the model can be found in [1, 10].

The main model components are ***activities*** and ***connectors***. The activities are associated with a context being defined as data passing to an activity. It is called ***input container***. An activity also returns data called ***output container***. Some output container elements of an activity can be passed to the input container elements of other activities or to the ***external memory***. All data elements are collected in the set **V.** Control and data connectors provide connections between the activities. A control connector has an associated Boolean predicate called ***transition condition***.

A directed graph based on sets of activities and control connectors is called ***control flow*** of workflow process. The execution of acyclic workflow process can be described in following general stages. At first, initial activities have to be executed and transition conditions of their outgoing control connectors should be calculated. Then, the set of ready activities have to be determined by the selecting activities whose all incoming transition conditions have been calculated. The most priority activity should be selected from this set, its join condition has to be calculated and the activity must be executed in the case of join condition truth-value. The execution of workflow process will be stopped if the set of ready activities is empty.

Let us extend the definition of a workflow processes to cover a cyclic workflow process [12]. An activity of the workflow process can be a cycle activity which can be a cycle head. The ***cycle head*** is the cycle entry point. The behavior of activities that are heads of cycles differs from the behavior of other activities. They have two different join conditions. The first join condition is made on control connectors that are connecting cycle activities with its head (***inner connectors***). The second join condition is made on control connectors that are connecting the cycle head with other activities (***outer connectors***). The cycle head executed in one of those join conditions would be satisfactory. A cycle activity, having at least one outgoing control connector with the cycle outside, is called a ***branching activity***.

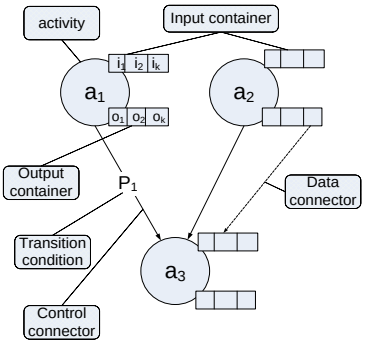

Figure 2: Workflow process graphical representation

Definition(process correctness) Let **P** be a workflow process. Let -**α, β** be two predicates depending on the state of the external memory **M** of **P**. Let us call **α** and **β** a precondition and a postcondition, respectively. In this case, **P** is correct with respect to **α** and **β,** if

$$[\ \alpha(M)=1 \Rightarrow \beta(MP)=1],$$

where **MP** denotes the external memory state after the execution of the **P** process.

2.2. Cycle Transformation Basic Operations

This section presents basic operations used by the transformation block (Figure 1). It is based on the idea of intervals suggested for program data flow analysis [8]. To implement cycle transformation during formal verification of an workflow process 4 basic operations are defined – interval set construction, replacement of each interval with an equivalent activity, interval condition construction for each interval branching activity, construction of interval transition conditions based on interval conditions. Basic transformation operation high level descriptions are listed bellow:

1. *Interval set construction(**construct_IntervalSet**)*;
    a. Find all intervals,
    b. Select intervals containing a cycle;
    c. Remove from the interval activities, that are not cycle activities;
2. *Replacement of each interval with an equivalent activity(**remove_Interval**)*,
    a. Remove cycle activities with their control and data connectors;
    b. Define new activity;
    c. Construct a new activity input container from input data elements of cycle activities connected with the cycle outside;
    d. Construct a new activity output container from output data elements of cycle activities connected with cycle outside and variables defined for the cycle branching activity states (0 means that the branching activity has not

been executed yet, *n* (*n* >= 1) means that branching activity has been executed *n* times.);

e. Add corresponding data connectors between the new activity and other activities for each removed data connector between the cycle activities and the cycle outside.

3. *Interval condition construction for each interval branching activity(**construct_IntervalCon**)*;
   a. Construct the set of branching activities that are supposed to be activated before the given branching activity;
   b. Construct the set of branching activities that have to be activated after the given branching activity;
   c. Build a branching activity condition to support two possible executions of a branching activity. The first one is the cycle execution termination without completing the cycle. The second one is all cycle activities execution at least once.
4. *Construction of interval transition conditions based on interval conditions(**construct_TransitionCon**);*
   a. For all branching activities construct interval conditions;
   b. Instead of control connectors connecting cycle head with the cycle outside, add control connectors connecting them with the new activity;
   c. Instead of control connectors connecting branching activities with the cycle outside, add control connectors connecting the new activity with those activities and formulate their transition conditions based on previous transition conditions and branching activity interval conditions;
   d. If the activity was connected with more than one branching activity, combine all transition conditions in one with an OR.

The mentioned operations formal description is presented in [12].

2.3. Mold Construction Basic Operations

This section presents basic operations used by the mold construction (Figure 1).

The ***workflow process state*** is a tuple composed by input variables of activities, transition conditions and external memory. ***Splitting set*** is a similar tuple whose components are a subset of corresponding variable domains. ***Splitting mold*** is a pair of two splitting sets. One of them is approximating the set of process current states from the lower bound, while the next one is approximating it from the upper bound. Let denote the set of all states of the workflow process by ***B*** and the set of all molds by ***M***.

To check correctness of the WFP correctness some basic operations are presented bellow

1. *Mold construction for precondition and postcondition* (***construct_PrePostMolds***);
   a. Select external memory variable contained in precondition/postcondition;
   b. Calculate the lower and upper approximations for each of them;
   c. For all other variables contained in the mold assign their domain sets.
2. *Define mold construction activity function* (***construct_ActMold***)
   a. Select the activity implementation function;
   b. Define the activity implementation function based on molds in such way;
      i. Select the lower/upper splitting set of the mold and construct its equivalent set of workflow process states;
      ii. Calculate the activity implementation application values on that process states;
      iii. Calculate lower/upper splitting set for the resulted process states;
3. *Define mold construction transition condition function*(***construct_TransMold***);
   a. Select the transition condition implementation function;
   b. Define the transition condition implementation function based on molds in such way;
      i. Select the lower/upper splitting set of the mold and construct its equivalent set of workflow process states;
      ii. Calculate the transition condition implementation application values on that process states;
      iii. Calculate lower/upper splitting set for the resulted process states;

The molds algebra and mentioned operations' formal description is presented in more details [10].

2.4. Transformation Conditions

Conditions are specified below for the checking if the cyclic workflow process is transformable to the acyclic one.

Transformation Conditions

1. The control graph is a first order factor graph [8].
2. Data flow connectors can connect an activity with another activity if they are connected in the control flow, too.
3. Branching activities, contained in a cycle, can have only two outgoing mutually exclusive transition conditions.
4. The memory elements can have single data connectors pointing onto them.

Let denote by ***check_transformabiltiy*** the corresponding operation and cycle existence operation by ***exist_cycles.***

2.5. Formal Verification Algorithm

Let us define the formal verification algorithm by the use of basing operations.

Algorithm V. Formal Verification Agorithm

Input Workflow process ***P,*** its precondition ***Pre*** and postcondition ***Post***

Output**:** One of values *{'**correct**', '**incorrect**' }*

Method:

1. If NOT ***exist_cycles*****(P)**, then
   - go to *step* 5
2. If NOT ***check_transformabiltiy*** **(P)**, then
   - go to *step 13*
3. $\mathbf{S_C \leftarrow}$ ***construct_IntervalSet*** **(P)**
4. for all $\mathbf{I(h) \in S_C}$, do
   a. construct the set of branching activities (BP):
   b. **P ←** ***remove_Interval*** **(P)**
   c. for all $\mathbf{br \in BP}$, do

d. add ***construct_IntervalCon***(**BP**) in **BrCond**

e. **P** ← ***construct_TransitionCon*** **(P, BrCond)**

5. **front** ←**{** $\mathbf{A_{start}}$**}**
6. ***construct_PrePostMolds***(**Pre, Post);**
7. for all **a** ∈**A** activity , do
   - ***construct_ActMold***(**a**)
8. for all **c** ∈**C** transition condition , do
   - ***construct_TransMold(c)***
9. **currMold** ← **mold(Pre)**
10. while **front** is not empty, do
    a. Select most priority activity **A**∈**front** and remove **A** from **front**.
    b. Calculate $\mathbf{mold_1}$ as a resulted mold for join condition of activity **A** (calculate it after calculating of all incoming transition conditions truth molds)
    c. Calculate $\mathbf{mold_0}$=¬ $\mathbf{mold_1}$
    d. If $\mathbf{mold_0}$ is not empty, then do
    - $\mathbf{finMold_0}$ ← Calculate the result on the **currMold**.
    Else, $\mathbf{finMold_0}$←Λ.
    e. If $\mathbf{mold_1}$ is not empty, then do
    - **nextMold** ←**(currMold) A**.
    For all **e** outgoing connectors of **A** *to* **A'** do i-iii)
       i. **mold(e)** ← (**nextMold) p**.
       ii. If all incoming connectors of **A'** have defined molds , then
       - insert **A'** into **front**
       iii. $\mathbf{finMold_1}$ ← Calculate the result on the **nextMold**.
    f. **finalMold** ← **finalMold** ∪**(** $\mathbf{finMold_0}$ ∪ $\mathbf{finMold_1}$**)**
11. If **(**upper splitting set of **finalMold) ⊆ (**lower splitting set of **mold(Post)**), then
    - **return (P** is correct with respect to **Pre** and **Post)**;
12. If **(**lower splitting set of **finalMold)⊄ (**upper splitting set of **mold(Post)**), then
    - **return (P** is incorrect with respect to **Pre** and **Post)**;
13. **END** (Simulation can be applied in this case to determine the WFP correctness [11,13])

The theoretical foundation of the basic operations used by the verification algorithm, their correctness and application domains are illustrated in details in [6, 7, 10 and 12].

3. CONCLUSION AND FUTURE DIRECTIONS

A formal verification algorithm of workflow processes, which is based on IBM/Workflow model, is suggested. The algorithm uses molds for upper and lower approximation of environment possible state sets. The consideration of mold transformations on all branching paths allows to avoid the process execution and examination per each input. The algorithm instead examines all branching paths through the molds. The weakness of the mold using technique is the possibility of getting uncertain answer to the question regarding the program correctness. Additional investigations on specific cases should be performed to estimate the probability of such a result.

It is planned to continue this work in two directions. One direction is to determine the range of the application of the suggested algorithm. The other direction is to improve the suggested verification algorithm, specifically, to increase the accuracy of the algorithm, to consider processes with fewer restrictions. The final goal is to find out limitations of the proposed approach.

REFERENCES

1. Leymann, F., Roller,D., "Production Workflow: Concepts and Techniques", *Prentice Hall Press* (2000)
2. IBM 2001. "IBM MQSeries Workflow: Concepts and Architecture", Version 3.3, GH12-6285-03, Product No. 5697-FM3, Mar. 2001, pp. iii-58.
3. Van der Aalst, W. M. P. and A.H.M. ter Hofstede, "Verification of workflow task structures: A petri-net-based approach", *Information Systems*, 25-1:43-69, 2000.
4. Perumal, S., Mahanti,A., "Cyclic Workflow Verification Algorithm For Workflow Graphs", *BPM and Workflow Handbook*, (2007)
5. Koehler,J., Hauser, R., "Untangling Unstructured Cyclic Flows; A Solution Based on Continuations", *CoopIS/DOA/ODBASE (1)*, pp. 121-138, 2004
6. Kostanian, A, "A Problem of Verification for Linear Programs with Boolean Arrays", *Kibernetika i sistemny analiz*, Kiev, Ukraine, 1995, #5, pp. 87-94.
7. Shoukourian, S.; A. Kostanian; V.Margarian and A.Ashour, "An Approach for System Tests Design and Its Applications", *Proc. of the 13-th IEEE VLSI Test Symposium*, NJ, Princeton, USA, 1995, pp. 448-453.
8. Allen,F.E., Cocke, J.: "A program dataflow analysis procedure", *Communications of the ACM*, 19(3):137-147 (March 1976)
9. P. Ammann and J. Offutt. "Introduction to Software Testing", *Cambridge University Press*, 2008. ISBN 978-0-521-88038-1
10. Kostanyan, A., Varosyan, A., "Partial Recognizing Algorithm for Verification of Workflow Processes". *Proceedings of "The Future Business Technology Conference"*, pp. 89-94, Porto, Portugal, (April 2008)
11. I. Boyakhchyan, A. Kostanyan, V. Matevosyan, S.Shoukourian, A. Varosyan "Tuning of IT Management Processes to a Computing Grid", *The Third International Conference of "Distributed Computing and Grid-technologies in Science and Education"*, June 30-July 4, 2008, Dubna, Russia
12. Kostanyan, A.; V. Matevosyan; S. Shoukourian; and A. Varosyan. 2009. "An Approach for Formal Verification of Business Processes". *Business and Industry Symposium (BIS'09)*, SpringSim'09, ACM Press, San Diego, CA, USA
13. I.Boyakhchyan, A.Kostanyan, V.Matevosyan, S.Shoukourian, A.Varosyan, "Process Based Management of Specific GRID Configurations: Verification of Changes", *Published in Proceedings of " The Future Business Technology Conference" (FUBUTEC'2009)*, EUROSIS, pp. 57-61, April 2009, Bruges, Belgium