Verification of Evidence Life Cycles in Workflow Diagrams with Passback Flows *

Osamu Takaki, Takahiro Seino, Izumi Takeuti, Noriaki Izumi and Koichi Takahashi
National Institute of Advanced Industrial Science and Technology (AIST)
2-41-6 Aomi, Koto-ku, Tokyo 135-0064, Japan
{o-takaki, seino-takahiro, takeuti.i, n.izumi, k.takahashi}@aist.go.jp

Abstract

We introduce the Evidence Verification Algorithm (EVA) in this paper, which verifies consistency of life cycles of evidences (evidence documents) in workflow diagrams. We used the AIST Workflow Language (AWL) as the syntax for workflow diagrams, which has additional information about evidences. A workflow diagram of AWL is essentially the same as a Unified Modeling Language (UML) activity diagram. EVA verifies the existence of consistent sequences of flows between the occurrences of evidences in a workflow diagram. It is important to verify consistency of life cycles of evidences, since some defects in the workflow diagram itself can be found by checking inconsistent life cycles of evidences in a workflow diagram.

Keywords: workflow diagram, verification, evidece life cycle

1. Introduction

We define a consistency property of life cycles of evidences described in a workflow diagram in this paper and introduce an algorithm that verifies the consistency property. Here "evidence" is a technical term which means an annotation on a workflow diagram, which denotes a document on which information is written, and/or with which something is approval, during the process of an operation. Workflow diagrams play a central role in describing business processes in the development of large-scale information systems, especially in analyzing the requirements of these systems. There have been several standard workflow languages such as BPMN [1] or XPDL [15], and numerous investigations into methodologies of verifying several consistency properties in workflow diagrams (e.g., see [12]).

*The authors are grateful to anonymous referees for their fruitful comments. This work was supported by 'Service Research Center Infrastructure Development Program 2008' from METI and Grant-in-Aid for Scientific Research (C) 20500045.

However, verifying consistency properties in the life cycles of documents, which are described in workflow diagrams, has not been sufficiently investigated. In large organizations such as large enterprises or governments, documents such as order forms, estimate sheets, specification descriptions, invoices, and receipts play significant roles for purposes of feasibility, accountability, traceability, or transparency of business. Tasks involve workers with different roles in such organizations, and these are carried out by circulating documents. Such documents are considered as kinds of evidences for the purposes above. We describe such documents with evidences in this paper. For simplicity, we often call such documents themselves "evidences".

Some evidences require office workers to carry out various tasks. Some evidences are manuals that teach workers how to conduct tasks. Some workers check evidences and sign them in when they accept their content. Therefore, numerous actual operations are currently based on evidences even if they are carried out with information systems. Consequently, it is important to consider workflow diagrams in which one can concretely and precisely describe the life cycles of evidences to analyze requirements in developing large-scale information systems.

When someone develops workflow diagrams, they often make errors in describing evidences, because their states are subtly affected by other evidences around them. Moreover, many inconsistencies in evidences come from inconsistencies in constructing the diagrams. The larger a diagram becomes, the harder it is to find inconsistencies in evidences that is in it.

However, verifying the consistency of evidence helps us to confirm correctness of the diagrams. In fact, we can find numerous defections and redundancies in flows by finding inconsistencies in evidences around the flows. Therefore, it is worth verifying evidences formally and/or automatically.

We define the consistency property of life cycles of evidences in a workflow diagram in this paper, which is described with a new language for workflow diagrams called AIST Workflow Language (AWL), and introduce the Evidence Verification Algorithm (EVA), which verifies the con-

sistency property of a given workflow diagram.

A workflow diagram of AWL is essentially the same as a Unified Modeling Language (UML) activity diagram. The control flows in the workflow diagrams of AWL as well as major workflow languages are described by arrows. Moreover, one can describe how to control evidences by using the arrows in the workflow diagrams of AWL. Since most operations are carried out with some specific evidences, it is rational for a single arrow to denote both a control flow and a flow of an evidence. By describing the flows of evidences with appropriate control flows, one can easily describe and understand the life cycles of the evidences. This paper targets AWL instead of UML, because AWL is used in the actual development of large-scale information systems at the National Institute of Advanced Industrial Science and Technology (AIST) [9].

Roughly, the life cycles of evidences mean a series of states of the evidences, and consistency of evidence life cycles in a workflow means that the workflow has no inconsistent life cycles of evidences. A consistent evidence life cycle is defined in a workflow diagram with correct structure, where "correct structureff is defined in [4] and [10].

EVA receives a workflow diagram as input data and returns a list of subgraphs as output data, each of which destroys a consistent evidence life cycle. EVA checks the "local evidence conditions", which we will define in this paper, to verify the consistency of evidence life cycles in a workflow diagram.

EVA is designed to verify life cycles of evidences in *acyclic* workflow diagrams. We introduce the Removing Algorithm of Passback Flows (RAPF), where a "passback flow" in a workflow diagram denotes a kind of flow that appears at the boundary of a "main stream" and the "replay of an operation" in the workflow diagram in order to apply EVA to cyclic workflow diagrams. We will define passback flows in a workflow by using the graph-theoretical properties of the workflow diagram, and define RAPF, which translates a workflow diagram into various workflow diagram(s) with no passback flows, which can be applied to EVA.

We implemented both EVA and RAPF, and used them to verify workflow diagrams in actual development at AIST. Although both EVA and RAPF target workflow diagrams of AWL, one can easily apply these algorithms to activity diagrams of UML.

This paper is based on a previous paper [7] but differs from it in three respects. First, we discuss the consistency property of life cycles of evidence and its verification algorithm based on workflow diagrams in AWL instead of UML activity diagrams. Second, we introduce RAPF to apply the verification algorithm to cyclic workflow diagrams. Third, we add proofs of the main lemmas, which we omitted from [7].

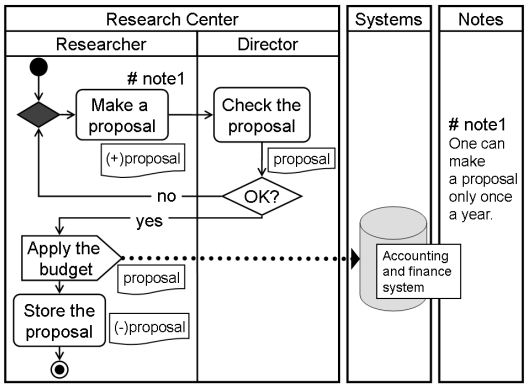

Figure 1. An example of a workflow diagram

The remainder of this paper is organized as follows. We introduce AWL in Section 2 and explain the consistency of life cycles of evidence in workflow diagrams in 3. We introduce EVA in Sections 4 and 6. We define local evidence conditions in Section 5 and introduce RAPF in Section 7. Section 8 explains the experimental results, by using an implementation of EVA. The experimental results indicate EVA could effectively be used to test and verify the consistency of life cycles of evidence to substantiate the construction of the workflow itself was consistent.

2. AIST workflow language

In this section, we explain a language of workflow diagrams [9], which is called "AWL" (AIST Workflow Language). AWL is defined to be appropriate to compose workflow diagrams for human workflow easily, and to verify consistency of evidence life cycles in workflow diagrams.

2.1. Overview of AWL

Comparing standard workflow languages such as BPMN or XPDL, the main feature of AWL is that in a workflow diagram of AWL one can assign to each activity a list of evidences (evidence documents) which are used in the activity.

The figure 1 is a workflow diagram describing a work of planning of a research. In the diagram, the rectangles and the pentacle denote operations needed for planning of a research. The figures near the polygons above denote evidences (evidence documents) used on the operations.

In this example, first a researcher composes a proposal of a research, and then a director checks the proposal. If the proposal passes the checking, then the proposal is returned to the researcher and he/she applies the budget on an accounting and finance system based on the proposal, and finally the proposal is stored by the researcher. If the

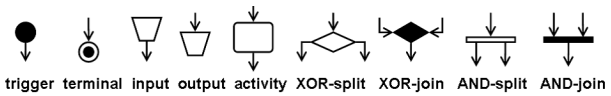

Figure 2. Shapes of nodes in workflow diagrams

proposal does not pass the checking, then the proposal is returned to the researcher and he/she remakes the proposal.

In this paper, we discuss only control flow and evidence life cycles in a workflow diagram. Therefore, we omit notions that are not relevant to control flow of workflows or evidences. For example, in this paper we do not consider data flows or actors in workflow diagrams.

2.2. Control flow of AWL

In the perspective of control-flow, workflow diagrams of AWL are similar to those in BPMN or XPDL, which are the most standard workflow languages, or workflows in previous researches such as [3], [5] or [13].

Definition 2.1 A workflow in the perspective of control-flow denotes a directed graph $W := (\mathbf{node}, \mathbf{flow})$ that satisfies the following properties.

1. $\mathbf{node}$ is a non-empty finite set, whose element is called a node in W.
2. $\mathbf{flow}$ is a non-empty finite set, whose element is called a flow in W. Each flow f is assigned to a node called a source of f and another node called a target of f.
3. Each node is distinguished, as follows: start, end, activity, XOR-split (branch), XOR-join (merge), AND-split (fork) and AND-join (rendezvous).
4. Whenever a flow f has a node x as the target (or the source) of f, x has f as an incoming-flow (resp. an outgoing-flow) of x. The numbers of incoming-flows and outgoing-flows of a node are determined by the type of the node. We itemize them in the following table.

	incoming-flows	outgoing-flows
start	0	1
end	1	0
activity	1	1
XOR-, AND-split	1	$\geqq 2$
XOR-, AND-join	$\geqq 2$	1

Table 1. Numbers of incoming- and outgoing-flows of a node

5. W has just one start and at least one end.
6. Activity diagrams are "simple" graphs, that is, for each activity diagram $\mathcal{A}$, and for each nodes N_1, N_2 in $\mathcal{A}$ there is *one* edge (flow) from N_1 and N_2 at most. Moreover, each activity diagram has *no circle edge*, that is, there is no flow from a node N to the same node N.
7. For a node x in W, there exists a path on W from the start node of W to x, where a path from s to x denotes a sequence $\pi = (f_0, \ldots, f_n)$ of flows in W such that the source of f_0 is s, the target of f_n is x and that the target of f_i is the source of f_{i+1} for each $i < n$. Moreover, there exists an end node e and another path on W from x to e.

2.3. Evidence

By "an evidence" in workflow diagrams one means a paper document or a data (a data file) of a document. In this paper, we regard an evidence as a paper document, which is composed, referred, re-written, judged, stored or dumped in some activities. Unlike data files, an evidence does not increase. Though one can make a copy of it, the copy is regarded not to be the same thing as the original evidence. Moreover, unlike data in a system multiple people can access simultaneously, an evidence can not be used by multiple people at the same time.

In formulating workflow diagrams, especially, those for human workflows, evidences are still very important even through a lot of paper documents are replaced by data (data files) in information systems. In a workflow diagram of AWL, evidences used in an activity is explicitly described in the activity, in order to describe and verify life cycles of evidences more correctly.

In the technical perspective, a list of evidences with length at least 0 is assigned to an activity, and an evidence E is defined to be a triple $(e, created, removed)$, where e is a label, and $created$ and $removed$ are boolean values. In what follows, we fix a non-empty set $\mathbb{E}$.

Definition 2.2 *Evidence* is a triple $(e, created, removed)$, where e is an element of $\mathbb{E}$ and $created$ and $removed$ are boolean values, that is, they are elements of $\{true, false\}$. For each evidence $E := (e, created, removed)$, we call e the *evidence label* of E.

Remark 2.3 In what follows, we denote E by the following ways.

(i) If $created = \text{false}$ and $removed = \text{false}$, then we abbreviate E to "e".

(ii) If $created = \text{false}$ and $removed = \text{true}$, then we abbreviate E to "$(-)e$".

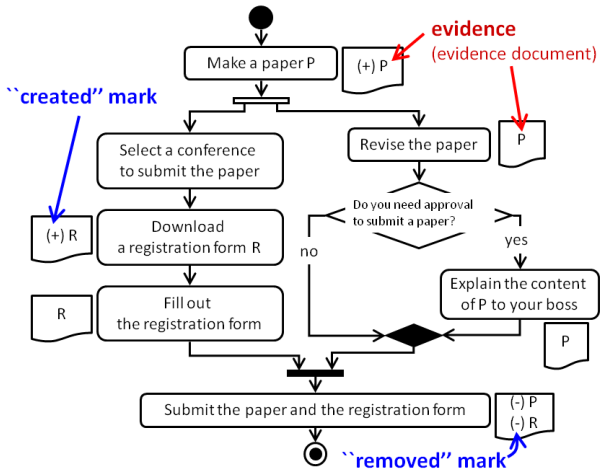

Figure 3. A workflow diagram of paper submission

(iii) If $created = \text{true}$ and $removed = \text{false}$, then we abbreviate E to "$(+)e$".

(iv) If $created = \text{true}$ and $removed = \text{true}$, then we abbreviate E to "$(+)(-)e$".

For a workflow diagram W, we consider an allocation which assigns to each activity in W a string of evidences. Note that such an allocation may assign to some activities the empty string, i.e., the string with length 0. By using workflow diagrams, one can express a lot of workflows. In order to explain evidences, we give an example of a workflow diagram which explains how to submit a paper, as follows.

For each workflow diagram W, each activity A in W and for each evidence E in the string assigned to A, we call E an evidence *on* A and call A an activity *having* E. Moreover, if the evidence label of E is denoted by e, we often call e the evidence label *on* A and call A an activity *having* e.

Moreover, for simplicity, we often identify each evidence with its evidence label. So, in what follows, *we often abbreviate "an evidence label" to "an evidence"*.

Remark 2.4 In what follows, we assume that, for each workflow diagram W and each activity A in W, A does not have multiple evidences sharing the same evidence label. We call the condition *the basic evidence condition*. [1]

Since each workflow diagram W is assume to satisfy the basic evidence condition, if an activity A in W has an evidence label e, A has just one evidence E with label e. So, we often say that e *is created (or removed) on* A if A has an evidence E having the $(+)$-mark (or the $(-)$-mark, respectively).

[1] We assume that each evidence can be differentiated from others even if some evidences share the same content. For example, if $\mathbf{E}$ is the set of documents, and if an evidence e is copied in an activity A, then one should not consider that A has two es, but should consider that A has e and a copy of e.

3. Consistency properties of workflow diagrams

The main subject of this paper is to verify consistency property of evidence life cycles in a workflow diagram. However, the consistency property is closely related to another consistency property of control flow of a workflow diagram. Thus, in this section, we first explain consistency property of control flow of a workflow diagram, and then we explain consistency property of evidence life cycles in the workflow diagram.

In the following subsections of this section and the three coming sections 4∼6, we will treat only *acyclic* workflow diagrams. That is, we assume that any workflow do not have a loop, where a loop denotes a sequence of flows

$$N_0 \xrightarrow{f_0} N_1 \xrightarrow{f_1} \cdots \xrightarrow{f_{n-1}} N_n \xrightarrow{f_n} N_0.$$

We will treat *cyclic* workflow diagrams in Section 7.

3.1. Correctness of Workflows

Consistency verification of workflows on the control flow perspective is one of the most important issues in research area of workflow verifications. There are a lot of researches of consistency properties of workflows in the viewpoint of control flow of them such as [2], [4], [5], [10], [11], [13] and [14].

An inconsistency of structures of workflows comes from a wrong combination of XOR-split/join nodes and AND-split/join nodes. Such inconsistencies are known as "deadlock" and "lack of synchronization" [5]. An acyclic workflow which is deadlock free and lack of synchronization free is said to be "correct" [13].

Definition 3.1 For an acyclic workflow W, an *instance* of W denotes a subgraph V of W that satisfies the following properties.

1. V contains just one start node. Moreover, for each node x in V, there exists a path on V from the start node to x.
2. If V contains an XOR-split c, then V contains just one outgoing-flow of c.
3. If V contains a node x other than XOR-split, then V contains all outgoing-flows of x.

Definition 3.2 Let W be an acyclic workflow.

1. An instance V of W is said to be *deadlock free* if, for every AND-join r in V, V contains all incoming-flows of r.

2. An instance V of W is said to be lack of *synchronization free* if, for every XOR-join m in V, V contains just one incoming-flow of m.

Definition 3.3 An acyclic workflow W is said to be *correct* if every instance V of W is deadlock free and lack of synchronization free.

Instances of a workflow diagram are not used not only to define correctness property but also to define consistency property of evidence life cycles in the workflow, which we will define in the next section.

3.2. Consistency property of evidence life cycles

Roughly, the "life cycle" of an evidence means that a series of states of the evidence. To be more exact, the life cycle of an evidence e (in $\mathbb{E}$) means when e is created, how e is moved to some activities, and when e is removed (archived or destroyed).

In order to define consistent life cycles of evidences in workflow diagram in a rigorous manner, we introduce some new concepts.

Since workflow diagrams have XOR-split nodes, one can regard each activity diagram $\mathcal{A}$ as an gathering of flow-sequences, each of which is obtained from $\mathcal{A}$ based on XOR-split nodes in $\mathcal{A}$. Based on the point, we introduce the following definition.

Definition 3.4 Let W be a workflow diagram and C the set of all XOR-splits on W. Then, a *phenomenon* on W denotes a function $\psi : C \to \mathbf{flow}(W)$ satisfying that $\psi(c)$ is an outgoing-flow of c for each $c \in C$, where $\mathbf{flow}(W)$ denotes the set of all flows in W.

Lemma 3.5 For a workflow diagram W and a phenomenon ψ on W, there exists a unique instance V of W such that for every XOR-split c in V the outgoing-flow of c in V is $\psi(c)$. We refer to the instance V as $W(\psi)$.

Conversely, for a instance V of W, there is a phenomenon ψ with $V = W(\psi)$.

The lemma above is easily shown.

One can consider each $W(\psi)$ as the workflow diagram obtained from W by extracting all activities and flows which take place under the phenomenon ψ.

Definition 3.6 For a workflow diagram W, a *line* in W is a sequence of flows in W

$$L = (A_1 \xrightarrow{f_1} A_2 \xrightarrow{f_2} \cdots \xrightarrow{f_{n-1}} A_n)$$

which satisfies the following properties.

(i) A_1 is an activity or the start in W.

(ii) A_n is an activity or an end in W.

(iii) $A_2, \ldots, A_{n-1}$ are nodes in W, each of that is not any activity, the start, nor any end.

For a line L above, A_1 is called the *source* of L, A_n the *target* of L and f_{n-1} the *target flow* of L.

Definition 3.7 A line L is said to be *equivalent* to another line L' if L and L' share the source and the target.

Definition 3.8 A sequence π of lines is said to be *equivalent* to another sequence π' of lines if there exist lines $L_1, \ldots, L_n$ and $L'_1, \ldots, L'_n$ such that

$$\pi = (A_1 \xrightarrow{L_1} A_2 \xrightarrow{L_2} \cdots \xrightarrow{L_{n-1}} A_n)$$

$$\pi' = (A_1 \xrightarrow{L'_1} A_2 \xrightarrow{L'_2} \cdots \xrightarrow{L'_{n-1}} A_n)$$

and that, for each $i = 1, \ldots, n$, L_i is equivalent to L'_i.

$L \sim L'$ (or $\pi \sim \pi'$) denotes that L is equivalent to L' (π is equivalent to π', respectively). Note that every line is equivalent to itself, and so is every sequence of lines.

Definition 3.9 Let W be an acyclic workflow diagram, ψ a phenomenon of W and let e be an evidence in W. Then, the *consistent life cycle* of e on $W(\psi)$ is the sequence π of lines in $W(\psi)$

$$\pi := (A_0 \xrightarrow{L_0} A_1 \xrightarrow{L_1} \cdots \xrightarrow{L_{n-1}} A_n)$$

which satisfies the following properties.
(i) Every activity A_i has e.
(ii) e is created on A_0.
(iii) e is not created on A_i for any i with $0 < i \leqq n$.
(iv) e is removed on A_n.
(v) e is not removed on A_i for any i with $i < n$.

Definition 3.10 An acyclic workflow diagram W is said to *have consistent evidence life cycles* if, for each phenomenon ψ of W, each activity A in $W(\psi)$ and for each evidence e on A, there is an essentially unique consistent life cycle π of e which contains A.

The statement "there is an essentially unique consistent life cycle π of e containing A" means that there is a consistent life cycle π of e containing A and that $\pi \sim \pi'$ for each consistent life cycle π' of e containing A.

4. Verification algorithm of evidence life cycles

The main body of this paper is to introduce an algorithm verifying consistency of evidence life cycles in a given acyclic workflow diagram, which is called *EVA* (Evidence Verification Algorithm).

Although details of EVA is described in the section 6, we here explain input and output data and the main property of EVA.

Input data of EVA: a correct workflow diagram W and all instances $\{W(\psi)\}$ of W.

Output data of EVA: a string consisting of pairs (L, e), where L denotes a sequence of flows and e an evidence.

Each (L, e) in output data denotes a defect of an evidence life cycle in input data.

Theorem 4.1 For each correct workflow diagram W, if the output data of W by EVA is the empty string, then W has consistent evidence life cycles, and vice versa.

This theorem is a corollary of Lemma 6.2 described in Section 6.

Remark 4.2 Here we omit explanations about how to extract instances from a workflow diagram and how to verify correctness of it. For details, see [6]. One can also refer to [2], [10], [11], [13] and [14],

5. Local evidence conditions

Actually, in order to verify consistency of evidence life cycles of a given correct workflow diagram W, EVA checks whether or not W satisfies "local evidence conditions". In order to explain there conditions, we first introduce a proposition and some definitions.

Proposition 5.1 For each workflow diagram W, each line L and for each evidence e contained in L, just one of the following properties holds.
(1) The source S and the target T of L share e, and e is not removed on S and e is not created on T.
(2) The source S and the target T of L share e, and e is removed on S, and e is created on T.
(3) The source S of L has e, e is removed on S, and the target of L does not have e.
(4) The target T of L has e, e is created on T, and the source of L does not have e.
(5) The target T of L has e and e is not created on T. Moreover, if the source S of L has e, then e is removed on S.
(6) The source S of L has e and e is not removed on S. Moreover, if the target T of L has e, then e is created on T.

One can easily show the proposition above.

Remark 5.2 Let L be an line with source S and target T activities, and let e be an evidence, for example, a document used in S or T or both of S and T. Moreover, assume that L contains no AND-split and no AND-join. Then, if S has e and e is not removed on S, e should exist on T. Therefore, (6) above can be regard as an wrong state. Similarly, if T has e and e is not created on T, e should "come from" S, and hence, (5) above can be regard as an wrong state as well as (6).

We next assume that L has an AND-split F and another line L' has S as its source. Then, when S has e, e is not removed on S, and when T does not have e, it is possible that e "pass" from S to the target of L'. In such a case, one can not assure that (6) is wrong. One can consider several similar cases.

Definition 5.3 A pair (L, e) of a line L and an evidence e is called a *line-evidence*.

Definition 5.4 To each line-evidence (L, e), we assign one of the following states.
(1) SCS (State of Consistent Succession) if (L, e) satisfies (1) in Proposition 5.1.
(2) SIR (State of Inconsistent Redundancy) if (L, e) satisfies (5) in Proposition 5.1.
(3) SID (State of Inconsistent Defection) if (L, e) satisfies (6) in Proposition 5.1.
(4) SCNS (State of Consistent Non-Succession) if (L, e) satisfies one of (2)$\sim$(4) in Proposition 5.1.

Definition 5.5 For a correct workflow diagram W, W is said to satisfies *local evidence conditions* if, for each phenomenon ψ of W, the restricted graph $W(\psi)$ satisfies the following conditions.
(1) For each line L in $W(\psi)$ and for each evidence e, if (L, e) is assigned SIR, then there exists a line L' sharing the target with L such that (L', e) is assigned SCS.
(2) For each line L in $W(\psi)$ and for each evidence e, if (L, e) is assigned SID, then there exists a line L' sharing the source with L such that (L', e) is assigned SCS.
(3) There are not two line-evidences (L, e) and (L', e), which are assigned SCS, and which share the source (or the target), but which do not share any node as their targets (or their sources, respectively).

The following lemma indicates that, for a workflow diagram W, if W is correct, local evidence conditions suffice to verify consistency of evidence life cycles in W.

Lemma 5.6 For each correct workflow diagram W, if W satisfies local evidence conditions, then W has consistent evidence life cycles, and vice versa.

We show this lemma in Appendix A of this paper.

6. Definition of EVA

By virtue of Lemma 5.6, in order to verify consistency of evidence life cycles of a given correct workflow diagram W, it is sufficient to check that W satisfies local evidence conditions. So, we establish EVA as an algorithm finding line-evidences (L, e) which violate local evidence conditions of a given correct workflow diagram.

Definition of EVA

(I) Input and output data of EVA are described in Section 4.

(II) The content of EVA is defined, as follows.
(EVA.1) Prepare three empty sets $\mathbf{Suc}$, $\mathbf{Inq}$ and $\mathbf{Res}$, which are provided for recording line-evidences.
(EVA.2) Execute EV.2.1 and EV.2.2 below in parallel.
(EVA.2.1) For a given W, starting the start of W, search all flows f in W by an appropriate graph search algorithm, [2] and all lines with target flow f.
(EVA.2.2) For each line L and for each evidence e contained in L, check the state of (L, e), and classify (L, e), as follows.

Case (i) where (L, e) is assigned SCS.

If L contains an AND-split or an AND-join, put (L, e) into $\mathbf{Suc}$. Otherwise, dump (L, e).

Case (ii) where (L, e) is assigned SIR.

If L contains an AND-join, put (L, e) into $\mathbf{Inq}$. Otherwise, put (L, e) into $\mathbf{Res}$.

Case (iii) where (L, e) is assigned SID.

If L contains an AND-split, put (L, e) into $\mathbf{Inq}$. Otherwise, put (L, e) into $\mathbf{Res}$.

Case (iv) where (L, e) is assigned SCNS.

Dump (L, e).

(EVA.3) For each phenomenon ψ of W, execute (EVA.3.1)$\sim$(EVA.3.3) below.
(EVA.3.1) Set $\mathbf{Suc}(\psi) := \mathbf{Suc} \cap W(\psi)$ and $\mathbf{Inq}(\psi) := \mathbf{Inq} \cap W(\psi)$, where $W(\psi)$ is regarded to be the set of line-evidences on $W(\psi)$.
(EVA.3.2) For each element (L, e) of $\mathbf{Inq}(\psi)$, check whether or not there exists an element (L', e) of $\mathbf{Suc}(\psi)$ such that (L, e) and (L', e) satisfy the property (1) or (2) in the definition of local evidence conditions. If (L, e) does not have such a (L', c), put (L, e) into $\mathbf{Res}$.
(EVA.3.3) Check whether or not there exist multiple elements of $\mathbf{Suc}(\psi)$ violating the property (3) in the definition of local evidence conditions. If there exist such elements, put them into $\mathbf{Res}$.
(EVA.4) Output $\mathbf{Res}$.

[2] We use a depth first search algorithm for implementation of EVA.

Remark 6.1 In most cases, it does not need to prepare all instances, since some instances $W(\psi), \ldots, W(\psi')$ share the same figure even though $\psi, \ldots, \psi'$ are not the same.

At the last, we show correctness of EVA.

Lemma 6.2 (Correctness of EVA) For each correct workflow diagram W, EVA terminates in finite steps. Moreover, all lines violating local evidence conditions of W are contained in the output of W by EVA, and vice versa. In particular, Theorem 4.1 holds.

We show this lemma in Appendix B.

7. Application of EVA to cyclic workflow diagrams

Until now, we have dealt with acyclic workflow diagrams. From now, we will discuss verification of correctness and consistency of evidence life cycles over cyclic workflow diagrams. In order to apply EVA to cyclic workflow diagrams, we extend the definitions of consistency properties in the previous sections to those over cyclic workflow diagrams. Thus, in order to extend the definitions of consistency properties, we consider a translation of cyclic workflow diagrams.

The main body of this section refers to [8].

Before defining the translation, we explain an observation of "real" workflow diagrams.

7.1. Observation of real workflow diagrams

By virtue of investigation of about 460 workflow diagrams, which have been composed in development of real large-scale information systems, we have the following observations about real workflow diagrams.

- **Observation 1.** Most loops in workflow diagrams contain flows which we call "passback flows".

A passback flow in a workflow diagram denotes a kind of a flow which appears at a boundary of a "main stream" and a "replay of an operation" in the workflow diagram. For example, in the figure 1, the main stream in the workflow is described by the sequence of flows between activities: "Make a proposal", "Check the proposal", "Apply the proposal" and "Store the proposal". On the other hand, if the proposal does not pass in the activity "Check the proposal", the proposal will be turned back to the previous activity "Make a proposal" via the flow labeled "no". In this case, the researcher have to replay the activity "Make a proposal". So, the flow "no" is a passback flow.

Workflow diagrams in the investigation above contain no loop which is considered in usual programs and assured its

termination property. That is, most loops in the workflow diagrams express replays of operations in the workflow diagrams. Thus, most loops contain passback flows.

- **Observation 2.** Every information described in a workflow diagram is treated as a "static" information. In particular, no information about operations after an operation is turned back via a passback flow.

For example, in the figure 1, in the case where a proposal is turned back to the activity "Make a proposal" via the passback flow "no", the researcher should not "make" a proposal, but "remake" the proposal. However, there is actually no modification for such a case in a workflow diagram.

It maybe possible to reconstruct such a workflow so that there is not such an inconsistency above which occurs on the target of the passback flow. Thus, one can select a policy which prohibit inconsistencies on the targets of passback flows like the example above. However, in many cases, this constraint is so strong that workflow diagrams have too large size or too complex structure. Therefore, in the real development, they do not select such a policy. Thus, we do not consider inconsistency between the target and the source of a passback flow in a workflow. Actually, from the real workflow diagrams in the investigation, we also have the following observation.

- **Observation 3.** The change of evidences during a passback flow is not described in the workflow diagram. In other words, even if there is some inconsistency between the evidences on the source node of (the first flow of) a path containing a passback flow and those on the target node of (the last flow of) the path, one should not regard it as an error.

In principle, evidences on each node in a workflow diagram are described only when a job stream arrived at the point for the first time. For example, the activity "Make a proposal" in the figure 1 has an evidence "(+)proposal", which is information described at the first time when the job stream arrives at the activity node from the start node. The evidence "(+)proposal" is not what is described in the case where a job stream arrives at the activity via the passback flow "no". Therefore, we should not consider that the evidence "proposal" in the activity "Check the proposal" changes to the evidence "(+)proposal" in the activity "Make a proposal" via the flow "no".

The observations above indicate that a verification algorithm should not output inconsistency between evidences on the source node of a path containing a passback flow and the target node of the path as an error. As a consequence of the discussion, we take the stance to deal with passback flows as special ones.

There maybe a workflow diagram in which a passback flow must not be regarded as a special one. We can not apply our methodology to such cases. We do not care about that, because such cases are rare.

7.2. Translation of cyclic workflow diagrams

In order to translate a cyclic workflow diagram into those to which EVA are applicable, we remove all passback flows in the cyclic. In this section, we first formalize passback flows in a workflow diagram, by using only graph-theoretical properties of the workflow diagram. Moreover, we introduce an algorithm which removes all passback flows in a given workflow diagram. For more details, see [8].

A path $(f_1, \ldots, f_n)$ in a workflow diagram W satisfying the following properties is called a *lariat path*.

1. The source of f_1 is a start node in W.
2. The target of f_n is the source of one of $f_2, \ldots, f_n$.
3. For each i and j with $i \neq j$, f_i and f_j do not share the same source.

The last flow f_n of a lariat path $\sigma := (f_1, \ldots, f_n)$ is called the *tail* of σ.

A path $(f_1, \ldots, f_n)$ in a workflow diagram W satisfying the following properties is called a *directly ending path*.

1. The source of f_1 is a start node in W.
2. The target of f_n is an end node in W.
3. For each i and j with $i \neq j$, f_i and f_j do not share the same source.

In the following two paragraphs, we consider what a passback flow is.

First, a passback flow heads in the opposite direction to a primary job stream and reaches such a job stream. Therefore, a passback flow is the tail of a lariat path.

Next, consider a flow contained in some directly ending path. Then, the flow is considered a member of a directly ending path, and the change of evidences during the flow is described in the workflow diagram, even if it is the tail of a lariat path. Therefore, the flow is not considered a passback flow.

By virtue of the discussion above, we can formalize passback flows, as follows.

Definition 7.1 A flow f in a workflow diagram W is called a *passback flow* if f is the tail of a lariat path in W and there is no directly ending path in W which contains f.

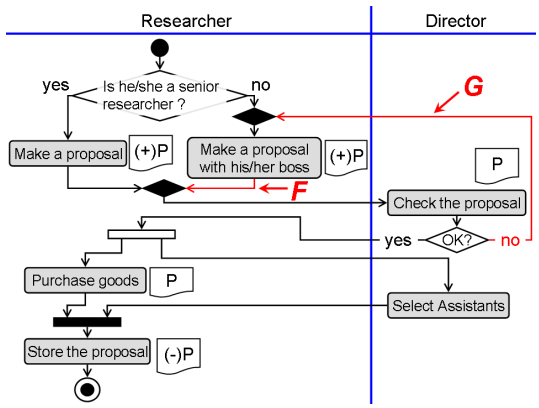

Figure 4. A workflow diagram of proposal submission

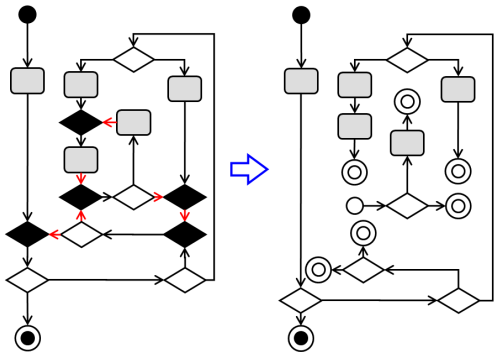

Figure 5. RAPF may divide a workflow diagram into multiple workflow diagrams

For example, in the figure 4, G is a passback flow, while F is not, even though F is the tail of a lariat path.

Now we define an algorithm *RAPF* (Removing Algorithm of Passback Flows) [8], which translates a workflow diagram W into some workflow diagram(s) with no passback flows, as follows.

1. Detect all flows in W and record tails $t_1, \ldots, t_n$ of all lariat paths in W.

2. Detect all directly ending paths and let $p_1, \ldots, p_m$ be tails each of which is not contained in any directly ending path. These tails are passback flows in W.

3. Replace targets $T_1, \ldots, T_m$ of $p_1, \ldots, p_m$ by new end nodes $E_1, \ldots, E_m$, respectively. These end nodes are called *additional end nodes*. Note that the number of incoming flows of each T_i decreases.

4. Execute the following operations corresponding to the number of incoming flows of each T_i. (Note that each T_i is an XOR-join or AND-join node.)

   (a) If T_i has more than or equal to two incoming flows, then leave T_i as it is.

   (b) If T_i has just one incoming flow, then replace the target of the incoming flow by the target of the outgoing flow of T_i, and then remove T_i as well as the outgoing flow of T_i.

   (c) If T_i has no incoming flow, then replace the source of the outgoing flow of T_i by a new start node and remove T_i. The new start node is called a *additional start node*.

RAPF makes no new lariat path. Moreover, any flow in a directly ending path does not become not to be contained in any directly ending path by RAPF. Thus, we have the following proposition.

Proposition. RAPF translates a workflow diagram W into RAPF(W), each element in which is a workflow diagram with no passback flows.

RAPF may output multiple workflow diagrams. We show an example in the figure 5. The operation on this example is executed mechanically by RAPF.

By RAPF, one can extend the definitions of correctness and consistency of evidence life cycles of acyclic workflow diagrams to those over cyclic workflow diagrams. In order to do so, we only have to redefine instances defined in Definition 3.1.

Definition 7.2 For an cyclic workflow W, an *instance* of W denotes a subgraph V of RAPFW that satisfies the properties 1)$\sim$3) in Definition 3.1.

By using the instances above, one can extend the definitions of correctness and consistency of evidence life cycles of acyclic workflow diagrams to those over cyclic workflow diagrams.

The role of RAPF is to extract a designer's intention of a workflow diagram. Here, the "designer's intention" is the intention which flows to be exceptional ones around which evidence life cycles should not be checked, and which flows to be usual ones around which evidence life cycles should be checked. Such a designer's intention is not completely formalizable, and hence, RAPF may not extract the designer's intention completely. However, from the observation of actual workflow diagrams, we claim that RAPF has enough ability to extract designers' intentions about passback flows correctly.

RAPF is not an algorithm which translates cyclic workflow diagrams to acyclic ones. Cyclic workflow diagrams are not contained in the range of EVA. Moreover, some

cyclic workflow diagrams are not translated into acyclic ones by RAPF. One can not apply EVA to such diagrams. However, such diagrams are rare, and we do not care about them.

8. Experimental results

The algorithm EVA is implemented as a java program called "evidenceVerifier", the input data of which is an xml-file expressing a workflow diagram, and the output data of which is an xml-file expressing a string of line-evidences which violate life cycles of evidences. [3] More properly speaking, evidenceVerifier has functions which execute the following for each input data W.
(1) verification of syntax and graph-theoretical properties (for example, connectivity) of W.
(2) translation of W to (an) acyclic workflow diagram(s) W^* by RAPF in Section 7.
(3) extraction of instances $\{S_1, \ldots, S_n\}$ of W^*.
(4) verification of basic and local evidence conditions of W^* based on $\{S_1, \ldots, S_n\}$.

In this paper, we omit explanations of algorithms which (1) and (3) are based on (one can refer to [6] for the algorithm of (3)).

By using the program, we verify a set of real workflows, which is a subset of the set of workflows used to design a real large enterprise application system of AIST. The subset consists of 60 workflows. Each of them has 5∼34 nodes and 5∼31 flows. All workflows are reviewed in a manual way in advance.

The output data of the workflows above by evidenceVerifier are classified into the following types.

(i)	*defects coming from complications of structures of workflows*	**6**
(ii)	*defects coming from inconsistencies of structures of workflows*	**8**
(iii)	defects by trivial mistakes	26
(iv)	superfluous error messages	10

Every execution time is within 0.5 second.

Important defects are those in (i) and (ii).

Defects in (i) come from unexpected flows of activities in workflows. For example, 2 defects in (i) come from a gap coming from changing flows of activities in a workflow in the later phases. It is difficult to find such defects by manual.

Defects in (ii) come from designers' essential misunderstanding on some activities or inconsistent structure of workflow diagrams. In particular, we could find 3 inconsistent flows of activities by the defections in (ii) above. This indicates that, in order to verify construction of a workflow, it is worth to verify consistency of evidence life cycles in it.

Defects in (iii) are defects which come from forgetting adding (+)-marks, forgetting removing (+)-marks, typos of evidence labels or forgetting describing evidences.

The error messages in (iv) consist of 3 messages by designers' informal omission and 7 messages by the relations to other workflow diagrams. As for designers' omission, designers sometimes omit some evidences on purpose. Our program does not corresponds to such omissions. As for the relations to other workflow diagrams, the program has not yet been implemented any function analyzing relationships between multiple workflow diagrams. For example, there is a workflow A which is a successive part of another workflow B, and an evidence E which is created in B and still occurs in A. Then the designer does not add (+)-mark to the first occurrence of E on A. However, our program outputs an error message to that. This is a superfluous error message.

9. Conclusion and future work

In this paper, we have defined a consistency property of evidence life cycles of a workflow diagram in AWL (AIST Workflow Language), and an algorithm EVA (Evidence Verification Algorithm), which verifies the consistency property of each workflow diagram. We also have defined local evidence conditions which are necessary and sufficient conditions for each workflow diagram to have consistent evidence life cycles (Definition 5.5 and Lemma 5.6). Moreover, we have shown that, for each correct workflow diagram W, EVA can determine whether or not W has consistent evidence life cycles (Theorem 4.1).

In Section 7, in order to apply EVA to cyclic workflow diagrams, we introduce an algorithm to translate them into acyclic ones. We first formalize a passback flow, which expresses (a boundary of a main stream and) a reply of an operation in a workflow. We formalize this flow with only graph-theoretical properties of the workflow diagram. We then give an algorithm RAPF, which detects and removes all passback flows in a given workflow diagram, and which translates cyclic workflow diagrams to acyclic ones, which EVA is applicable to.

In Section 8, we experimented with an implementation "evidenceVerifier" of EVA and 60 workflow diagrams of a real large enterprise application system, and have shown that evidenceVerifier could find 38 defects of evidence life cycles of the workflow diagrams and that we could find several defects of structure of the workflow diagrams from the defects of evidence life cycles of them.

As a future work, we are developing the way to verify

[3] Properly speaking, input data express workflows corresponding to workflow diagrams. Moreover, output data have several additional informations.

other kinds of life cycles of workflow diagrams, for example, life cycles of operations and data in databases which are described on activities.

References

[1] Business Process Management Initiative (BPMI). *Business Process Modeling Notation (BPMN) Version 1.0*. Technical report, BPMI.org, 2004.

[2] H. Lin, Z. Zhao, H. Li, and Z. Chen. A novel graph reduction algorithm to identify structural conflicts. In *Proceedings of the 35th Annual Hawaii International Conference on System Science (HICSS)*. IEEE Computer Society Press, 2002.

[3] R. Liu and A. Kumar. An analysis and taxonomy of unstructured workflows. In *Proceedings of 3rd International Conference on Business Process Management (BPM)*, LNCS 3649, pages 268–284. Springer, 2005.

[4] W. Sadiq and M. E. Orlowska. On correctness issues in conceptual modeling of workflows. In *Proceedings of the 5th European Conference on Information Systems (ECIS)*, pages 943–964, 1997.

[5] W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117–134, 2000.

[6] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Algorithms verifying phenomena independence and abstracting phenomena subgraphs of UML activity diagrams. In *Software Engineering Saizensen 2007*, pages 153–164. Kindaikagaku-sha (in Japanese), 2007.

[7] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Verification algorithm of evidence life cycles in extended UML activity diagrams. In *Proceedings of The 2nd International Conference on Software Engineering Advances (ICSEA 2007)*. IEEE Computer Society Press, 2007.

[8] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Quality improvement of workflow diagrams based on passback flow consistency. In *Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008)*, pages 351–359. INSTICC, 2008.

[9] O. Takaki, T. Seino, I. Takeuti, N. Izumi, and K. Takahashi. Workflow diagrams based on evidence life cycles. In *Proceedings of the 8th Joint Conference on Knowledge - Based Software Engineering 2008 (JCKBSE 2008), Frontiers in Artificial Intelligence and Applications*, pages 145–154. IOS Press, 2008.

[10] W. M. P. van der Aalst. Verification of workflow nets. In *Application and Theory of Petri Nets 1997*, LNCS 1248, pages 407–426. Springer, 1997.

[11] W. M. P. van der Aalst. The application of petri nets to workflow management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[12] W. M. P. van der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management. In *Lectures on Concurrency and Petri Nets*, LNCS 3098, pages 1–65. Springer, 2004.

[13] W. M. P. van der Aalst, A. Hirnschall, and H. M. W. Verbeek. An alternative way to analyze workflow graphs. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE)*, LNCS 2348, pages 535–552. Springer, 2002.

[14] H. M. W. Verbeek, T. Basten, and W. M. P. van der Aalst. Diagnosing workflow processes using woflan. *The Computer Journal*, 44(4):246–279, 2001.

[15] Workflow Management Coalition (WfMC). *Workflow Management Coalition Workflow Standard: Workflow Process Definition Interface - XML Process Definition Language (XPDL)*. (WfMC-TC-1025), Technical report, Workflow Management Coalition, Lighthouse Point, Florida, USA, 2002.

A. Proof of Lemma 5.6

In this subsection, we first show that, for a correct workflow diagram W, if W satisfies local evidence conditions, then W has consistent evidence life cycles, and then we show the converse proposition.

Here we consider only an acyclic workflow diagrams.

Definition A.1 For an acyclic workflow diagram W and a node n in W, the *degree* of n in W denotes the maximum of the lengths of the paths from the start node of W to n. Moreover, the *height* of n in W denotes the maximum of the lengths of the paths from n to some end node of W.

Let W be a correct workflow diagram, ψ a phenomenon of W, A an activity in $W(\psi)$ and let e be an evidence in A.

Now we construct a consistent evidence life cycles F of e on $W(\psi)$

$$F := (A_0 \xrightarrow{L_0} \cdot\cdot \xrightarrow{L_{n-1}} A_n = A = B_0 \xrightarrow{R_0} \cdots \xrightarrow{R_{m-1}} B_m),$$

where $L_n, \ldots, L_0, R_0, \ldots, R_{m-1}$ and R_m are lines.

Claim 1. There is an essentially unique sequence of lines $A_0 \xrightarrow{L_0} \cdot\cdot \xrightarrow{L_{n-1}} A_n = A$ which satisfies the following properties

(i) Every activity A_i has e.

(ii) e is created on A_0.

(iii) e is not created on A_i for any i with $i > 0$.

(iv) e is not removed on A_i for any i with $i < n$.

Proof of Claim 1. We construct $A_0 \xrightarrow{L_0} \cdot\cdot \xrightarrow{L_{n-1}} A_n$, by using induction on the degree of A in W.

(1) If e is created on A, then we set A_0 to be A.

(2) Assume that e is not created on A. Then, by local evidence conditions, there exists an essentially unique line L with target A. Moreover, the source S of L contains e,

which is not removed on S, and the degree of S is less than that of A. Therefore, by induction hypothesis, there is an essentially unique sequence of lines $S_0 \rightarrow \cdots \rightarrow S_k = S$. Thus, we obtain the desired sequence $S_0 \rightarrow \cdots \rightarrow S_k \xrightarrow{L} A$. Moreover, by the definition of equivalence relation on sequences of lines, the desired sequence is essentially unique. □

Claim 2. There is an essentially unique sequence of lines $A = B_0 \xrightarrow{R_0} \cdots \xrightarrow{R_{m-1}} B_m$ which satisfies the following properties

(i) Every activity B_i has e.

(ii) e is removed on B_m.

(iii) e is not created on B_i for any i with $i > 0$.

(iv) e is not removed on B_i for any i with $i < m$.

Proof of Claim 2. One can show the claim in the similar way to the proof of Claim 1, by using induction on the height of A in W. □

By Claims 1 and 2, we obtain an essentially unique consistent life cycle of e in $W(\psi)$. Therefore, we have shown that if W satisfies local evidence conditions, then W has consistent evidence life cycles.

Now we assume that W has consistent evidence life cycles and show that W satisfies local evidence conditions. That is, we show that $W(\psi)$ satisfies the properties (1)∼(3) in Definition 5.5.

(1) Let L be a line in $W(\psi)$ and (L, e) a SIR line-evidence. Then, the target T contains e but e is not created on T. Thus, by consistency of evidence life cycles in W, there exists an essentially unique consistent evidence life cycle

$$A_0 \rightarrow \cdots \rightarrow A_{k-1} \xrightarrow{L_{k-1}} A_k = T \rightarrow \cdots \rightarrow A_n$$

with $k > 0$. Since A_{k-1} contains e but e is not removed on A_{k-1}, (L_{k-1}, e) is a SCS line-evidence. We have the result.

One can show the properties (2) and (3) in the similar way to the proof of (1) above. □

B. Proof of Lemma 6.2

Since it is clear that EVA terminates in finite steps, we show that, for a given correct workflow diagram W, the output of W by EVA is the set of all line-evidences which violate local evidence conditions of W.

We first remark that the division of line-evidences in the step EVA.2.2 in the definition of EVA does not depend on instances, since the state of a line-evidence (L, e) is determined only by the state of E on the source and the target of L.

We now show that every line-evidence which violates the property (1), (2) or (3) in Definition 5.5 is contained in the set $\mathbf{Res}$ at the step (EVA.4) in the definition of EVA. Let $W(\psi)$ be an instance of W and (L, e) a line-evidence in $W(\psi)$.

(1) If (L, e) violates the property (1) in Definition 5.5, (L, e) is assigned SIR, but there exists no SCS line-evidence (L', e) such that L' is contained in $W(\psi)$ and shares the target with L. Thus, if L contains no AND-join, (L, e) is put into $\mathbf{Res}$ in the step (EVA.2.2).(ii). If L contains an AND-join, then (L, e) is put into $\mathbf{Inq}$ in the step (EVA.2.2).(ii), and moved to $\mathbf{Res}$ in the step (EVA.3.2). So, (L, e) is contained in $\mathbf{Res}$ in the step (EVA.4).

(2) If (L, e) violates the property (2) in Definition 5.5, one can obtain the same result in the similar way to (1) above.

(3) If (L, e) violates the property (3) in Definition 5.5, (L, e) is assigned SCS, and there exists another SCS line-evidence (L', e) such that L' is also contained in $W(\psi)$ and shares the target or source with L. Thus, (L, e) and (L', e) are put into $\mathbf{Suc}$ in the step (EVA.2.2).(i), and moved to $\mathbf{Res}$ in the step (EVA.3.3). So, (L, e) is contained in $\mathbf{Res}$ in the step (EVA.4).

We finally show that every line-evidence in $\mathbf{Res}$ violates the property (1), (2) or (3) in Definition 5.5. Let (L, e) be a line-evidence in $\mathbf{Res}$. Then, it is put in $\mathbf{Res}$ in the step (EVA.2.2.(ii)), (EVA.2.2.(iii)), (EVA.3.2) or (EVA.3.3).

(1) If (L, e) is put in $\mathbf{Res}$ in the step (EVA.2.2.(ii)), (L, e) is assigned SIR but L contains no AND-join. Thus, for any instance $W(\psi)$ which contains L, there is no line which shares the same target with L, since L contains no AND-join. So, (L, e) violates the property (1) in Definition 5.5.

(2) If (L, e) is put in $\mathbf{Res}$ in in the step (EVA.2.2.(iii)), one can show that (L, e) violates the property (2) in Definition 5.5 in the similar way to (1) above.

(3) It is clear that every line-evidence which is put into (EVA.3.2) or (EVA.3.3) violates one of the properties in Definition 5.5.

Thus, we have completed the proof of Lemma 6.2. □