Philippine Journal of Science
152 (6B): 2327-2342, December 2023
ISSN 0031 - 7683
Date Received: 23 May 2023

Classical Soundness in Robustness Diagram with Loop and Time Controls

Jasmine A. Malinao[1]* and Richelle Ann B. Juayong[2]

[1]Division of Natural Sciences and Mathematics,
University of the Philippines Tacloban College, Tacloban City, Leyte 6500 Philippines
[2]Department of Computer Science, College of Engineering,
University of the Philippines Diliman, Diliman, Quezon City 1101 Philippines

This study introduces and formalizes the concept of classical soundness for robustness diagrams and loop controls. This concept has not been established and formalized in previous literature for this type of workflow model. Although this concept may vary in syntax across different types of workflows (*e.g.* workflow nets, business process modeling and notation, Petri nets), this study adopts the well-known overarching semantic of classical soundness, which requires that every activity or work done by a system that is determinable from its workflow model observes proper termination and liveness. In the meantime, we discount reset-bound subsystems in our analysis of the classical soundness of our target model. We define classical soundness, along with the formalism and design strategies, to achieve proper termination and liveness on this model in both structural and behavioral aspects of the activities therein. Finally, we establish and prove the time and space complexity of proving classical soundness under these aspects for this model.

Keywords: model verification, soundness, workflows

INTRODUCTION

Robustness diagram with loop and time controls (RDLT) – as well as its predecessor, robustness diagram of the unified modeling language (UML) – are workflow models that have been used to represent real-world systems such as integrated disease surveillance and response (Lopez *et al.* 2020), heating, ventilation, and air-conditioning systems (Malinao 2017; Malinao *et al.* 2016; Rezk 2012), and Fujitsu Ten's computer-aided multi-analysis system auto test tool (Malinao *et al.* 2013). Along with the introduction and use of RDLT in various specializations came its model properties and verification strategies (Sulla and Malinao 2023*;* Yiu *et al.* 2018; delos Reyes *et al.* 2018; Malinao 2017). An advantage to using the RDLT in modeling systems over others is its capability to provide a unified view of all workflow dimensions (van der Aalst 1996) – namely, resource, process, and case.

While indeed there are other models like RDLTs that use all three workflow dimensions in system representation, *e.g.* business process modeling and notation (BPMN), activity diagrams, *etc.*, there are still existing gaps and challenges in using and verifying them. Some of these gaps and challenges pertain to concept excess, lack of support for explicit representation of data and rules, problems in process orchestrations and information hiding, and functional decomposition *vis-à-vis* model complexity (Leopold *et al.* 2015; Stiehl 2014; Weske 2007). Meanwhile, RDLTs provide an unambiguous set of components while offering freedom on model expressiveness and familiarity such that they can still be decomposed and/or mapped to mathematical models (Petri nets, workflow nets, matrix representation, and

*Corresponding author: jamalinao1@up.edu.ph

related procedures) or to some well-known uni- or multi-dimensional UML diagrams, as demonstrated in previous literature works (Yiu *et al.* 2018; delos Reyes *et al.* 2018). With the recognition of the rich literature on workflow verification of various modeling frameworks, our work builds on top of this corpus to formulate model properties, *e.g.* soundness, free-choiceness, well-handledness, *etc.*, for RDLT, albeit with cognizance to RDLT's inherent multi-dimensionality in system representation. Past RDLT literature (Malinao 2017) has already introduced similar model properties and verification strategies thereof.

Arguably one of the most important properties that had been formulated for RDLTs is the concept of relaxed soundness (Malinao 2017; Sulla and Malinao 2023). Soundness has different notions in the literature of workflow nets (van der Aalst *et al.* 2011; van der Aalst 1996) and, apart from proper termination and liveness, would imply or relate to various properties such as boundedness, safeness, well-handledness, *etc.* The most stringent among the notions of soundness would be classical soundness (van der Aalst *et al*. 2011), as shown in Figure 1. The arrows signify implication, *e.g.* classical soundness of a workflow net also implies that it is also relaxed, easy, weak, and lazy sound. The latter notions of soundness would put on different levels of relaxations of proper termination and liveness – with classical soundness requiring both. With cognizance of the importance of formulating soundness in models and the absence of such a definition of this property for RDLTs, it is the goal of this paper to address this gap in RDLT and, in general, in multidimensional modeling and verification. Along with this definition, we would also establish the structural and behavioral requirements of RDLTs that can be used to verify it in tractable amounts of resources. Note that we shall limit this study to RDLTs without their reset structures and capability in relation to these profiles.

PRELIMINARIES

In this section, we describe the preliminaries needed for understanding our investigated model. Nevertheless, it is our assumption that the reader has a basic knowledge of graph theory and workflow nets (Cormen *et al.* 2009; Quiwa 2007; van der Aalst 1996).

Definition 1: RDLT (Malinao 2017). An RDLT is a graph representation R of a system that is defined as $R = (V, E, T, M)$, where:

- V is a finite set of vertices, where every vertex is either a boundary, entity object, or controller.
- E is a finite set of arcs where no two objects are connected to each other. Furthermore, every arc (x, y) has the following attributes:
  - $C: E \rightarrow \Sigma \cup \{\epsilon\}$, where Σ is a finite non-empty set of symbols, and ϵ is the empty string. $C(x, y) \in \Sigma$ means that $C(x, y)$ is a condition that is required to be satisfied, *e.g.* input/output requirement, to proceed from x to y. Meanwhile, $C(x, y) = \epsilon$ means that there is no condition imposed by (x, y) or signifies that x is the owner object of the controller y.
  - $L: E \rightarrow \mathbb{N}$ is the maximum number of traversals allowed on the arc.
- $T: E \times \mathbb{N}^n$ is a mapping such that $T((x, y)) = [t_1, \ldots, t_n]$ for every $(x, y) \in E$, where $n = L((x, y))$ and $t_i \in \mathbb{N}$ is the time a check or traversal is done on (x, y) by some algorithm's walk on R.
- $M : V \rightarrow \{0,1\}$ indicates whether $u \in V$ is the center of a reset-bound subsystem (RBS). Given a center u ∈ V, where M(u) = 1, an RBS is a subgraph G_u of R that is induced by u and its set of owned controllers.

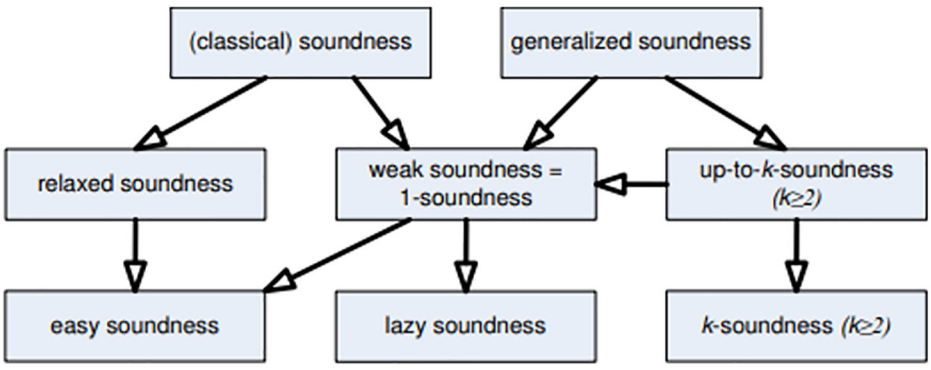

Figure 1. The different notions of soundness, as presented by van der Aalst *et al*. (2011).

An arc (x, y) ∈ E is said to be a bridge of G_u if and only if [1] x is not a vertex in G_u but y is. We then say that (x, y) is an in-bridge of y in G_u or [2] x is a vertex in G_u but y is not. We then say that (x, y) is an out-bridge of y in G_u. A pair of arcs (a, b) and (c, d) are type-alike (with respect to y) if and only if [1] y is present in both arcs and [2] either both arcs are bridges of y of some G_u or both are not.

A vertex x ∈ V is called a source(sink) if there are no incoming(outgoing) arcs incident to (from) *x*. Also, we use *diam(R)* to denote the diameter of the underlying graph of *R*.

Figure 2 shows an RDLT *R* that has two entity objects x_1 and y_1, where $\{x_2, x_3, x_4, x_5\}$ and $\{y_2,y_3\}$ are their owned controllers, respectively. x_1 and y_3 are the source and sink of *R*, respectively. Furthermore, *R* also has an RBS, with w_1 as its center and its set of owned controllers $\{w_2, w_3\}$. For w_1, the type-alike arc sets are $\{(x_4, w_1), (x_5, w_1)\}$ and $\{(w_1, w_2), (w_1, w_3), (w_3, w_1)\}$. The label *c: v* on every $(x, y) \in E$ of *R* shows $C(x, y)$: $L(x, y)$. The notation $[t_1, t_2, \ldots, t_n]$ on (x, y) records the time of check, traversal, or a reset done on (x, y). The sequence $[t_1, t_2, \ldots, t_n]$ can be dynamically updated by traversal algorithms, *e.g.* the activity extraction in the study by Malinao (2017), used on the RDLT.

Given an RDLT R, the activity extraction algorithm in the study by Malinao (2017) is used to generate an activity done by the system being represented by *R*. This activity *S* is composed of the objects (*e.g.* manpower) and controllers (*e.g.* tasks), as well as a set of cases that involve them. It generates this activity using an input vertex (*i.e.* source) and its corresponding output vertex (*i.e.* sink) in *R*. Formally, $S = \{S(1), S(2), \ldots, S(k)\}$, where each reachability configuration *S*[i] is a set of arcs traversed by the algorithm at time *I*, $1 \leq i \leq k$. The algorithm dynamically updates the *T*-attributes of the arcs to realize each *S(i)*.

From the source vertex in *V* of *R*, the aforementioned algorithm iterates on checking, traversals, backtracks, or resets using some conditions until it reaches the sink in *R*. In particular, the algorithm starts with *R* having its *T*-attributes as zero vectors. For every iteration of the algorithm, it chooses in a non-deterministic manner an incident arc (x, y) of the currently-reached vertex *x*. It performs a check on (x, y), where it compares the *T*-values of (x, y) against its *L*-value. If the number of nonzero values of $T(x, y)$ is smaller than $L(x, y)$ (the check is successful), then traversal on (x, y) is possible; otherwise, not. If the check is successful, then the algorithm updates $T(x, y)$ with a time of the check, *i.e.* $t + 1$, where *t* is the most recent time of traversal from among the incoming arcs of *x*. Otherwise, the algorithm considers (x, y'), where $y' \neq y$ for activity extraction.

Upon a successful check on (x, y), the algorithm shall then evaluate if (x, y) is unconstrained (by Definition 2) with respect to every type-alike (v, y) ∈ E. If (x, y) is unconstrained, the algorithm traverses (x, y). This means the algorithm visits *y* and considers all checked *(v, y)*. The check time in *T(v, y)* is replaced with the largest among all the incoming, checked arcs of *y.* This also means that every such *(v, y)* is also unconstrained, ergo, traversed at this time; otherwise, it considers other incident arcs of *x* for checking. If no such arcs remain, then the algorithm backtracks to a parent of *x* and proceeds with checking from there. No two successive checks can be done on *(x, y)*. If *(x, y)* is traversed and *(x, y)* is an out-bridge of an RBS in *R,* then all the *T*-attributes of the arcs inside the RBS are reset as zero vectors.

Definition 2: unconstrained arc (Malinao 2017). Let $R = (V, E, T, M)$ be an RDLT. An arc $(x, y) \in E$ is unconstrained if $\in (v, y) \in E$, where (x, y) and (v, y) are type-alike, any of the following requirements hold:

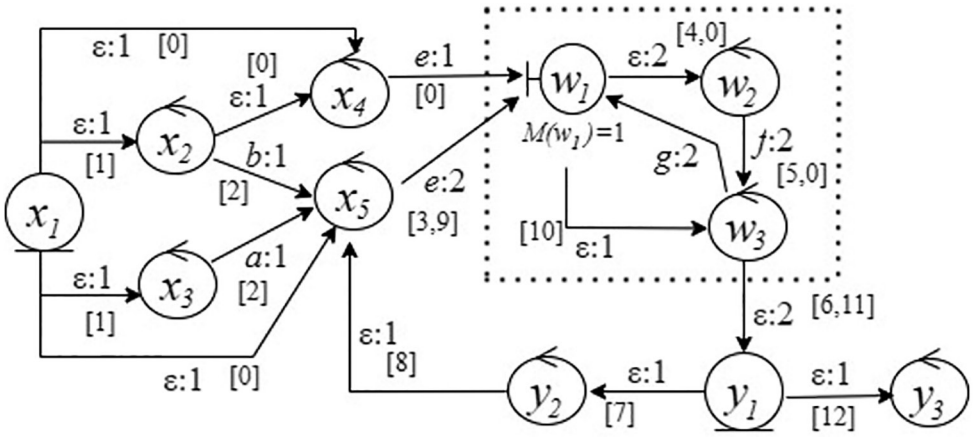

Figure 2. RDLT with a reset-bound subsystem with center at w_1, where $M(w_1) = 1$. RDLT with a reset-bound subsystem with center at w_1, where $M(w_1) = 1$. The *T*-values of the arcs in this figure are already marked with the time of traversals of the algorithm on *R.*

1) $C(v, y) \in \{\epsilon, C(x, y)\}$, *i.e.* the condition of (v, y) is the same as (x, y), or there is no condition imposed by (v, y) at all.
1) $|\{t_i \in T(x, y) | t_i \geq 1\}| \leq |\{t_j \in T(v, y) | t_j \geq 1\}| \leq L(v, y)$, *i.e.* the number of checks and traversals done on (x, y) has not exceeded that of (v, y)'s.
2) $C(v, y) \in \Sigma$, $C(x, y) = \epsilon$ and $T(v, y) \neq [0]$, *i.e.* there is no unsatisfied constraint $C(v, y)$. That is, (v, y) has already been checked and/or traversed by the algorithm.

The RDLT in Figure 2 shows an activity S, where $S(1) = \{(x_1, x_2), (x_1, x_3)\}$, $S(2) = \{(x_2, x_5), (x_3, x_5)\}$, …, $S(11) = \{(w_3,y_1)\}$, $S(12) = \{(y_1,y_3)\}$ of the source x_1 and sink y_3.

Definition 3: maximal substructure (Malinao 2017). Given a set of source vertices $I \in V$ of the RDLT R and a sink vertex $o \in V$, a support $V' \in V$ of $i \in I$ and o is a set of vertices, where $x \in V'$ is both a descendant of I and an ancestor of o. A maximal support $V'' \in V$ of $i \in V$ and o is a support of i and o, where there is no support V' of i and o, where $V' \in V''$ in R. A maximal substructure V''' is the union of all the maximal support of all $I \in I$ and o.

Definition 4: looping arc (Malinao 2017). Given an RDLT R, a vertex $x \in V$, the antecedent set $\alpha(x)$ is a set of vertices found in at least one elementary path in R from a source $w \in V$ to vertex x. Meanwhile, the consequent set $\Omega(x)$ is another set, where $\Omega(x) = \{v \in V \mid v$ is a descendant of x along a path from x to u, $u \in \alpha(x)$, v $\in \alpha(x)\}$. An arc $(a, b) \in E$ is a looping arc of x if [1] vertex $a \in \Omega(x)$ and vertex $b \in \alpha(x)$, or [2] vertex $b \in \alpha(x)$ and vertex $a = x$ if $\Omega(x) = \in$.

With these, we have the following structures for RDLTs:

1. **AND-JOIN at $y \in V$.** This structure is composed of distinct type-alike arcs $(v, y), (u, y) \in$ E having different non-empty string conditions, *i.e.* $C(v, y) \neq C(u, y) \neq \epsilon$.

2. **MIX-JOIN at $y \in V$.** This structure is composed of at least one pair of distinct type-alike arcs (v, y), $(u, y) \in$ E, where $C(v, y) \in \Sigma$, and $C(u, y) = \epsilon$.

3. **OR-JOIN at $y \in V$.** This structure is composed of type-alike arcs $(v, y), (u, y) \in E$ having the same conditions, *i.e.* $C(v, y) = C(u, y)$.

In Figure 2, $\{(x_1, x_5), (x_2, x_5), (x_3, x_5), (y_2, x_5)\}$ forms an instance of a MIX-JOIN at x_5 in R. If we remove $\{(x_1, x_5), (y_2, x_5)\}$ from R, $\{(x_2, x_5), (x_3, x_5)\}$ forms an AND-JOIN merging at x_5. The arc sets $\{(x_4, w_1), (x_5, w_1)\}$ and $\{(x_1, x_4), (x_2, x_4)\}$ are examples of OR-JOINs. Since the activity extraction algorithm chooses non-deterministically from x to any of its adjacent vertex y in its sequential exploration of R, this means that splits (*e.g.* OR-split, XOR-split, parallel split) are not discriminated at the point of the split in R. Nonetheless, these splits can be designed and implemented in RDLTs together with the aforementioned JOINs, where the paths emanating from the split would merge.

When the algorithm reaches one of the components, *e.g.* *(x, y)*, of an AND- or MIX-JOIN *(x, y)* and *(v, y)*, where *(x, y)* is deemed constrained, we say that the JOIN is resolved if and when *(v, y)* is unconstrained (so too is *(x, y)*) at a subsequent evaluation of the algorithm; otherwise, the JOIN is unresolved.

RELAXED SOUNDNESS IN RDLT

In the context of Petri and workflow nets, classical soundness requires that a model exhibits [1] proper termination for each of the cases in the given net, and [2] liveness, *i.e.* every task in the net is reachable and can be executed in at least one of these cases. Proper termination requires that whenever the final component of the net is reached, *i.e.* its process completes, there are no other processes that have yet to be completed in the net. Meanwhile, liveness ensures that there are no deadlocks or livelocks that prevent the case completion or termination.

Although classical soundness might be an ideal property for a workflow to satisfy, it might be unnecessary to represent real-world systems. For example, proper termination for every case can be waived as long as liveness is present. Such are the requirements of relaxed soundness in workflow nets. That is, relaxed soundness allows some processes to not complete if one completes and triggers termination of the net. Workflow nets that are relaxed sound are allowed to have potential deadlocks and livelocks, for as long as there is at least one process that completes.

Definition 5: activity profile (Malinao 2017). Given an RDLT R, a set of source vertices $I \in V$ and a sink vertex $f \in V$, an activity profile $S = \{S(1), S(2), ..., S(k)\}$, $1 \leq k \leq diam(R)$, associated with set I and vertex f have arcs incident from a source vertex in $S(1)$ and an arc incident to the final output vertex in $S(k)$. Formally, $(x, y) \in S(1)$ for each $x \in I$, and an arc $(u, f) \in S(k)$ for some vertices $y, u \in V$.

In the context of RDLTs, relaxed soundness is defined as follows.

Definition 6: relaxed soundness of RDLTs (Malinao 2017; Sulla and Malinao 2023). An RDLT R is a relaxed sound if, for every sink $f \in V$ and a source $w \in R$, where w is an ancestor of f, there exists an activity profile $S = \{S(1), S(2), \ldots, S(k)\}$, where $1 \leq k \leq diam(R)$, where the following conditions hold:

i. For every reachability configuration *S(t)* prior to *S(k)*, if any, there should be at least one arc *(x, y)* ∈ *S(t)* and another arc *(y, z)* ∈ *S(t')*, for a time *t'*, where $t < t' \leq k$. This ensures that there is at least one continuous path from *w* to *f* in activity *S*.

ii. The final reachability configuration *S(k)* is only composed of arcs that point to the sink *f, i.e.* ∀(*x, y*) ∈ *S*(*k*), *y* = *f*.

iii. The set of arcs traversed at a time *t* is strictly contained in the set of arcs traversed at time *t+1*, *i.e.* $\cup_{i=1}^{t} S[i] \subset \cup_{i=1}^{t+1} S[i]$. Since an arc can occur in multiple *S*[i], the operator ∪ should be considered a multiset union operator.

iv. Every arc *(x, y)* ∈ *E* is traversed in some activity profile *S', i.e.* there exists an activity profile *S'*, where (*x, y*) ∈ *S'*(*t*) for some *S'*(*t*) ∈ *S'*.

Definition 6 follows the requirements for relaxed soundness in workflow nets (van der Aalst *et al*. 2011). Furthermore, by requirements [i], [iii], and [iv] of Definition 6, we know that every MIX-JOIN or AND-JOIN structure is resolved in at least one (terminating) activity in *R*. Moreover, by requirements [ii] and [iii], we know that every exploration of the algorithm to construct an activity does not yield a deadlock nor livelock (as that of workflow nets), *i.e.* the algorithm cannot proceed toward the sink/final vertex of *R*, nor indefinitely stagnate at a repeating reachability configuration. Therefore, for each arc (*x, y*) ∈ *E*, the algorithm can perform a series of "good" (non-deterministic) choices for checks and traversals along a set of paths that involve (*x, y*), where each JOIN therein is resolved until the target sink vertex is reached.

ESTABLISHING CLASSICAL SOUNDNESS IN RDLTS AND ITS RELATED PROFILES

From here on, we will be establishing definitions and strategies that are no longer part of previous works in literature. Before providing the details of the contributions of this paper, we summarize the methodological approach of this paper as follows:

1. Define classical soundness in the context of RDLTs.

   As there are many other properties that can be ascertained for classical sound workflow nets, we aim to achieve in this work a contextually equivalent notion of classical soundness for RDLTs, accounting for its multidimensionality, however discounting the presence of RBSs at the moment.

2. Define the concept of criticality of arcs, in particular, relating to the impact of their *L*-attributes and reusability in all the activities that are derivable from *R*.

3. Define structural requirements of JOINs in RDLTs, with a focus on the arc attributes of the JOIN components, that help achieve classical soundness in *R*.

4. Build theorems (and proofs) that establish the relationships between the structural requirements above and the classical soundness of RDLTs.

5. Measure the complexity of verifying classical soundness in RDLTs.

6. Provide conjectures on scaling the results above to RDLTs with RBS.

Definition 7: classical sound R. An RDLT *R* is classical sound if for every activity profile *S* = {*S*(1), *S*(2), ..., *S*(*k*)}, $1 \leq k \leq diam(R)$, *diam*(*R*) is the diameter of *R*, of a set of source vertices $I \subset V$ and a final output vertex $f \in V$, where ∀*x* ∈ *I* and *y* ∈ *V*, (*x, y*) ∈ *S*(1), and (*u, f*) ∈ *S*(*k*), *u* ∈ *V*, the following hold.

1. **Proper termination.** For every activity profile *S* = {*S*(1), *S*(2), ..., *S*(*k*)}, $1 \leq k \leq diam(R)$, of a set of source vertices $I \subset V$ and a final output vertex $f \in V$.

   - All arcs in the final reachability configuration *S(k)* must be incident to sink *f*, *i.e.* for every (*x, y*) ∈ *S*(*k*), *y* = *f*.
   - If k ≥ 2: every arc incident to a vertex *y* in a reachability configuration *S[i]* has a corresponding arc incident from vertex *y* in a succeeding reachability configuration *S(j), i.e.* for every (*x, y*) ∈ *S*[i], there exists another arc (*y, z*) ∈ *S*(*j*), for all $1 \leq i < k$, and for some *j* in the range $i + 1 \leq j \leq k$.

2. **Liveness.** Every arc is traversed in some activity profile, *i.e.* for every (*x, y*) ∈ *E*, there is an activity profile *S'* = {*S'*(1), *S'*(2), …, *S'*(*k'*)}, where (*x, y*) ∈ *S'*[i], $1 \leq i \leq k'$.

Definition 7 under Item 1 requires that every vertex *y* that is reached from *x* and included in the activity must have its subsequent child vertex *z*, if any, be also included in the activity and their processes eventually end up at the target output vertex *f* – thus, JOINs would always be resolved. Furthermore, Item 1 requires that every vertex that is reached during the termination step is the target output vertex f of the activity. This means that there is no unfinished process at the termination of the activity. Both these requirements compose proper termination of *R*. Lastly, Item 2 requires that every arc is included in at least one activity in *R*, ensuring no deadlocks nor livelocks in *R*.

PROFILES CAUSING BACKTRACKING AND UNFINISHED PROCESSES

As a start, we analyze cases where backtracking happens in the activity extraction algorithm. We illustrate these cases using Figure 3 below. For simplicity purposes, we only use type-alike vertices in the subsequent discussions, without loss of the applicability of our proposed algorithms on R with or without RBSs.

1. Influence of conditions (*i.e. C*-values) on unfinished processes. The activity profile S extracted from the RDLT in Figure 3 shows the unresolved AND-join (v, q), $(w, q) \in E$ when the activity extraction algorithm terminated upon reaching y. That is, a backtrack was done by the algorithm upon checking (v, q) since it is not unconstrained due to (w, q) (note that traversal time is enclosed in square brackets while check time is enclosed with a parenthesis in the figure). With these, the RDLT in Figure 3 does not satisfy proper termination; albeit, it is live. Meanwhile, it can also be shown that R in Figure 3 is a relaxed sound.

The combination of AND-joins and the non-deterministic choice of the algorithm in its exploration of paths/processes should indeed be accounted for, among other things, when attempting to achieve classical soundness in RDLTs. This also holds for MIXED-joins such as $\{(q, y), (x, y)\}$ in Figure 3.

2. Influence of *L*-values on unfinished processes. A check of the algorithm on (x, y) considers whether a traversal is (still) feasible on it by comparing its L- and T-values. Whenever this check is unsuccessful, it backtracks to find other processes from x. We can find a similar dilemma in processes that are not finished in the RDLT in Figure 3. This can happen when we traverse (x, o) again, *i.e.* the algorithm cannot proceed to traverse (w, o) because we have fully maximized its L-value. Hence, the algorithm backtracks to x and goes to y. Note that the activity produced here has a final reachability configuration $S(k) = \{(x, o), (x, y)\}$ for some $k \in \mathbb{N}$. This means that the algorithm terminated, having reached y, with the (subsequent) task/process involving o unfinished.

INFLUENCE OF LOOPS IN THE ACTIVITIES OF *R*

The reachability of vertices in an RDLT relies on the configuration of the L-values of the arcs that connect them. In previous literature, relaxed soundness requires that the traversals from a vertex x to its children are at most the entry traversals of x from its parents to control a possible looping back of the algorithm to the ancestors of x. This looping back can trigger deadlocks. Meanwhile, classical soundness, this neighborhood, along with others, needs to be accounted for while dealing with every process that leads (back) to x.

Remark 1. From here on, we shall only consider and state looping arcs as those looping arcs of every $x \in V$ in R, where there is no $r \in \Omega(x)$ connecting to $s \in \alpha(x)$.

Definition 8: critical arc, escape arc. A cycle $c = [x_1 x_2 \ldots x_n]$ is a sequence of vertices $x_i \in V$, where $(x_i, x_{i+1}) \in E$, $i = 1, 2, \ldots, n-1$, and no two vertices in c are the same except for $x_1 = x_n$. The elements of c are denoted as $Lit(c)$. We denote the set of arcs of c as $ArcsOfCycle(c) = \{(x, y) \in E | \exists x_i, x_{i+1} \in Lit(c)$, where $x = x_i$ and $y = x_{i+1}\}$.

$(x, y) \in ArcsOfCycle(c)$ is called a critical arc(CA) in c if it has the minimum L-value among the arcs in c, *i.e.* $L(x, y) = min_{(u, v) \in ArcsOfCycle(c)}\{L(u, v)\}$. If $\exists (x,v) \in E$ such that $v \in V \setminus Lit(c)$, then we refer to (x,v) as an escape arc(EA) of (x, y) in c. Self-loops, *i.e.* (x, x), are cycles that are themselves entirely composed of one critical arc that does not affect the (re)use of other arcs in an RDLT.

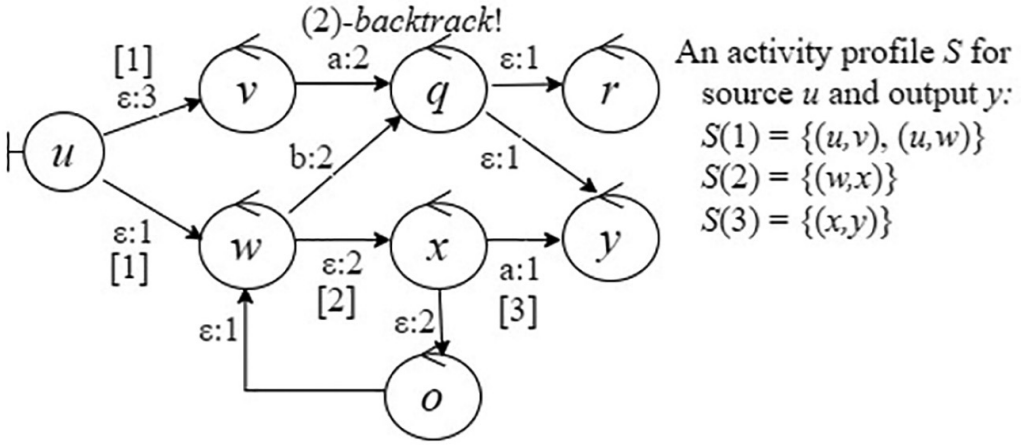

Figure 3. RDLT with an activity S and a pending task q with an unresolved AND-join.

For the RDLT in Figure 4, the vertex x_3 has its set of looping arcs $\{(x_6, x_3), (x_9, x_7)\}$. For discussion purposes, let the cycles be $c_1 = [x_3x_4x_6x_3]$, $c_2 = [x_3x_5x_6x_3]$, $c_3 = [x_3x_4x_6x_8x_9x_7x_3]$, $c_4 = [x_3x_5x_6x_8x_9x_7x_3]$, $c_5 = [x_7x_8x_9x_7]$, and $c_6 = [x_8, x_8]$. For c_1, its CA = (x_6, x_3), EA = $\{(x_6, x_8), (x_6, x_{10})\}$, and other critical arc/s(OCA) of another cycle c_j intersecting this cycle (OCA) is null. Meanwhile, for c_3, its CA = (x_7, x_3), EA = $\{(x_7, x_8)\}$, OCA = $\{(x_8, x_9)\}$.

For simplicity of our discussions, we shall first focus on the connectivity, *L*-values of arcs, and the cycles they are involved in *R* of Figure 4. Subsequently, we continue with dealing with JOIN structures to achieve classical soundness in *R*.

NON-CRITICAL ARCS AND THEIR REUSABILITY

If an NCA $(x, y) \in E$ is not involved in any cycle in *R*, then it cannot appear multiple times in any activity of *R*, *e.g.* (x_1, x_2), (x_6, x_{10}) in Figure 4. Otherwise, we set a "good" choice for $L(x, y)$ that helps achieve the classical soundness of *R.* That is, it always allows the reuse of (x, y) through any and all of the cycles it is involved in if such cycles exist. Specifically, we take the sum of the distinct CAs that are found in cycles that involve (x, y), considering the CA with the least *L*-value in the set of CAs found within each cycle. If this is achieved for (x, y), then we refer to (x, y) as a loop-safe arc.

Definition 9: loop-safe arc. Let *R* be a connected RDLT, *i.e.* $\forall v \in V$, there is a path from a source $s \in V$ to v in *R*. A NCA $(x, y) \in E$ of *R* is loop-safe if $L(x, y) > RU(x, y)$, where:

$$RU(x, y) = \sum_{c_k=1}^{|Cycles(x,y)|}(J * L(u, v)), \text{ for some } (u, v) \in minL\_CA(c_k), \text{ with} \quad (1)$$

$$J = \begin{cases} 1, & \text{if } k = 1 \text{ or } \bigcup_{j=1}^{k-1} minL\_CA(c_j) \cap minL\_CA(c_k) \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

and $minL\_CA(c) \subset E$ is the set of arcs whose *L*-value is the minimum among the CAs found in *c*, accounting for the other cycles *c′* in *R* that intersect with *c*.

If there is no cycle involving (x, y), *i.e.* $Cycles(x, y) = \phi$, this means that (x, y) is never reused in any activity in *R*, *i.e.* $RU(x, y) = 0$; therefore, simply set $L(x, y) \in \mathbb{N}$.

For example, suppose we set $L(x_9, x_7)$ of *R* in Figure 4 to help achieve classical soundness for *R*. In particular, we consider the reusability of (x_9, x_7) in *R*. (x_9, x_7) is involved in cycles c_3, c_4, and c_5 but not in c_1, c_2, and c_6. The algorithm can explore from x_1 and use the ancestors of x_9(and x_7), including those not involved in the former set of cycles to reach (x_9, x_7). However, it needs to engage first in any of these cycles for (x_9, x_7)'s inclusion in an activity. With these, we have the following notes:

1. For cycle c_3, its set of CAs (including those from other cycles) is $\{(x_8, x_9), (x_7, x_3)\}$ with $\{L(x_8, x_9) = 3, L(x_7, x_3) = 2\}$. When traversing through c_3, the reuse of (x_9, x_7) is influenced by the CA that has the minimum *L*-value, *i.e.* $L(x_7, x_3) = 2$, among all CAs that are found in this cycle, *i.e.* (x_8, x_9) of c_5. That is, (x_9, x_7) can be reused through c_3 in at most 2 times, as per $L(x_7, x_3) = 2$, rather than of $L(x_8, x_9) = 3$.

2. For cycle c_4, we find the same scenario as with the reusability of (x_9, x_7) with respect to c_3. With c_4, the reuse of (x_9, x_7) is influenced by $L(x_7, x_3) = 2$.

3. For cycle c_5, there is only one CA, *i.e.* (x_8, x_9) with $L(x_8, x_9) = 3$, and none from other cycles intersecting with c_5. With c_5, the reuse of (x_9, x_7) is influenced by $L(x_8, x_9) = 3$.

4. Note that we only consider the distinct Cas, which impacts the reuse of (x_9, x_7) to decide on the goodness of $L(x_9, x_7)$, *e.g.* (x_7, x_3) is considered for

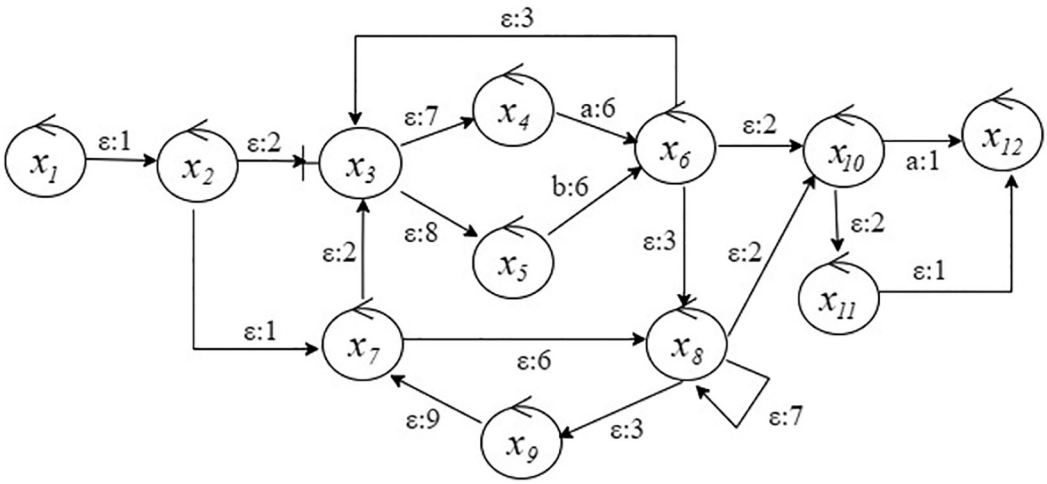

Figure 4. RDLT with cycles, critical arcs, and escape arcs.

both c_3 and c_4 above. Furthermore, these CAs would supersede each other based on their *L*-values whenever they are found to co-occur in one cycle, *i.e.* we take those having the minimum *L*-value.

5. From the evaluations above, a good value for $L(x_9, x_7)$ to help achieve classical soundness in *R* would be greater than 2 + 3 = 5, *i.e.* $L(x_7, x_3)+(x_8, x_9)$. Since $L(x_9, x_7) = 9 > 5$ in *R* of Figure 4, then the current L-value of (x_9, x_7) helps satisfy classical soundness in *R*.

The NCA (x_9, x_7) is influenced by cycles c_3, c_4, c_5. With this, by Definition 9, $minL\_CA(c_3)$, $minL\_CA(c_4)$, and $minL\_CA(c_5)$ are $\{(x_7, x_3)\}$, $\{(x_7, x_3)\}$, and $\{(x_8, x_9)\}$, respectively. Thus, $RU(x_9, x_7) = (1 * 2)+ (0 * 2)+ (1 * 3) = 5$. Because $L(x_9, x_7) = 9 > 5$, therefore, (x_9, x_7) is loop-safe. Meanwhile, $RU(x_{11}, x_{12}) = 0$.

Remark 2. For the subsequent discussions, we shall focus and – with full intent – only pertain to RDLTs with no RBS. This is because RBS requires additional specifications to ensure classical soundness in RDLTs. Nevertheless, we provide some brief insights as to how we handle RDLTs with RBS in Appendix III.

CRITICAL ARCS AND ESCAPE ARCS

Since we already know good choices for *L*-values for NCAs, we now deal with CAs themselves, as they may cause undesirable backtracks by the algorithm. For this, we focus on their neighborhood – rather than their *L*-values – to prevent unfinished processes. This is because their *L*-values are recommended to be independent of other *L*-values of other arcs in *R* to facilitate control of the (re)use of RDLT components.

Consider the cycle $c = [w\ x\ o]$ of Figure 3, whose CA is (o, w) with $L(o, w) = 1$. Note when *o* is reached again, the algorithm can never traverse (o, w) again. With this, the process from *o* to *w* is unfinished, and a backtrack to *x* is incurred, eventually reaching the sink *y*. Thus, $S(6) = \{(x, o), (x, y)\}$ violates the first requirement of classical soundness for RDLTs.

In order to avoid the above violation, we ensure that there is at least one EA (x,z) for every CA (x, y) in every cycle of *R*, if $\exists(u, x) \in E$. We also require that such EAs are loop-safe. (x,z) should not be a CA since there is no assurance of its reusability when the algorithm fails to traverse (x, y). (*R* in Figure 3 has only one CA (o, w) that has no EA.)

Definition 10: safe critical arc. A CA $(x, y) \in E$ is safe if there is an escape, non-critical arc $(x,z) \in$ E, where (x,z) is loop-safe.

With the cycles and arc profiles of *R* in Figure 4, we realize that *R* has safe CAs.

ON JOINS, L-VALUES, AND REUSABILITY

We now discuss the requirements and design strategies for JOINs that help achieve classical soundness in RDLTs, regardless of backtracks made by the algorithm.

Definition 11: JOIN-safe L-values. For every pair of arcs $(u, y), (v, y) \in E$, where $C(u, y) \neq C(v, y)$, we say that (u, y) and (v, y) have JOIN-safe *L*-values if:

1. **One split origin.** There is exactly one common ancestor $x \in V$ for *u* and *v* such that there is exactly one path $P_u = a_1a_2...a_n$, where $a_1 = x$, $a_{n-1} = u$, $a_n = y$, $a_i \in V$, $1 < I < n-2$, and another path $P_v = b_1b_2...b_m$, where $b_1 = x$, $b_{m-1} = v$, $b_m = y$, $b_j \in V$, $1 < j < m-2$. Furthermore, P_u and P_v do not intersect with each other except at *x* and *y*. That is, no a_i and b_j along these paths are equal, for $1 < i < n$, $1 < j < m$; and

2. **No unrelated process.** For every arc $(a, b) \in E$, if $a = x$, where *x* is the one common ancestor of *u* and *v*, then there is no path from a to another vertex $r \in V$ such that for some $(r,s) \in E$, $s \neq y$.

3. **No branching out from every related process.** $\forall q_k \in Lit(P_q)$, where $q \in \{u, v\}$, $1 \leq k < |Lit(P_q)|$, $\nexists(q_{k,s}) \in E$ where $s \in V \setminus Lit(P_q)$.

4. **No process interruptions.** $\forall q_k \in Lit(P_q)$, where $q \in \{u, v\}$, $1 < k \leq |Lit(P_q)|$, $\nexists(s, q_k) \in E$, where $s \in V \setminus Lit(P_q)$.

5. **Duplicate conditions**

   f. If $C(u, y), C(v, y) \in \Sigma$, *i.e.* an AND-JOIN at *y*, then there is no process $P = [x_1x_2...x_p]$ in *R*, where $x_1 = x$, $x_p = y$, such that $C(x_{p-1}, x_p) = C(u, y)$ (or $C(v, y)$).

   g. Without any loss of generality, if $C(u, y) = \epsilon$ and $C(v, y) \in \Sigma$, then any process $P = x_1x_2...x_p$ in *R*, where $x_1 = x$, $x_p = y$, can have $C(x_{p-1}, x_p) \in \{\epsilon, C(v, y)\}$. That is, a MIX-JOIN can have duplicate conditions for the arcs connecting to y, but no two of such arcs have different conditions.

6. **Equal L-values of arcs at the AND-JOIN.**

7. **Loop-safe components of every related process.** For an AND- or MIX-JOIN merging at *y*, each of its processes $P = x_1x_2...x_k$, $k \in \mathbb{N}$, where $x_1 = x$ and $x_k = y$, the arc $(x_i, x_{i+1}) \in E$, $i = 1, 2, \ldots, k-1$, is loop-safe. Lastly, for an OR-JOIN merging at y, each process $P = x_1x_2\ldots x_k$, $k \in \mathbb{N}$, of this JOIN has its arc $(x_i,$

x_{i+1}) to have a JOIN-safe L-value, i = 1, 2, …, k – 1, if (x_i, x_{i+1}) is either a loop-safe arc or a safe CA in R.

Definition 12: JOIN-safe R, L-safe R. A RDLT *R* is JOIN-safe if every JOIN in *R* has its set of arcs to have JOIN-safe *L*-values. An RDLT *R* is *L*-safe if every NCA is loop-safe, every CA is safe, and *R* is JOIN-safe.

For *R* in Figure 4, we have: [a] AND-JOIN at x_6 (split at x_3) with processes $P_{x4} = x_3x_4x_6$ and $P_{x5} = x_3x_5x_6$, with this AND-JOIN being JOIN-safe; [b] MIX-JOIN at x_{10} (split at x_{12}) with processes $P_{x10} = x_{10}x_{12}$ and $P_{x11} = x_{10}x_{11}x_{12}$, with this MIX-JOIN being JOIN-safe; and [3] the rest of the JOINs in *R* are OR-JOINs that are JOIN-safe. Note that all its NCAs and CAs are loop-safe and safe, respectively. Since we have also shown that *R* is JOIN-safe; therefore, *R* is *L*-safe.

MAIN FINDINGS ON CLASSICAL SOUNDNESS OF RDLT

Shown below are the theorems that we can establish from the structural and behavioral profiles of *R* in relation to classical soundness, as well as the asymptotic analysis in verifying them. The proofs of these theorems are in Appendices I and II, respectively. Whenever *R* contains multiple sources and/or sinks, we construct its extended RDLT *R'* (Malinao 2017), where a dummy source *i*' and dummy sink *o'* are added to *V* to build vertex set *V'* of *R'*. Furthermore, let the arc set *E'* of *R'* to be equal to *E* of *R* and (i', i), $(o', o) \in E'$, for every source $i \in V'$ and sink $o \in V'$, with $L(i', i) = L(o', o) = 1$, $C(i', i) = \epsilon$, and $C(o', o) \in \Sigma$. Note that the *L*- and *C*-values are carried over from *E* to *E'*.

Theorem 1. An RDLT *R* with a single source $s \in V$ and single sink vertex $o \in V$ is classical sound if *R* is *L*-safe.

Theorem 2. The extended RDLT *R'* of a multi-sink *R* is classical sound if *R'* is *L*-safe.

Theorem 3. Let *R''* be the maximal substructure of a sink vertex $o \in V$ and its set of source vertices $I \subset V$ of a RDLT *R*. Let *R'''* be the extended RDLT of *R''*.

Lemma 1. The time and space complexity to verify the loop-safeness of *R* is $O(c|E|^4)$ and $O(c\,|E|)$, respectively, where *c* is the number of cycles in *R*.

Lemma 2. Given an RDLT *R* that has a loop-safe set of NCAs, the time and space complexity of verifying that *R* has safe CAs is $O(|E|^2)$ and $O(|V| + |E|)$, respectively.

Lemma 3. Given an RDLT *R* that has a loop-safe set of NCAs and safe CAs, the time and space complexity of verifying that *R* is JOIN-safe is $O(|V|\ |E|^2)$ and $O(|E|^2)$, respectively.

Theorem 4. Verifying that an RDLT *R* is *L*-safe takes $O(c|E|^4)$ and $O(|E|^2)$ time and space complexity, respectively.

Corollary 1. Verifying that an RDLT *R* is classical sound takes $O(c|E|^4)$ and $O(|E|^2)$ time and space complexity, respectively.

CONCLUSIONS AND FUTURE WORK

This research has introduced and formalized classical soundness for RDLTs. Furthermore, the structural and behavioral profiles accounting component reusability that help achieve classical soundness in RDLTs with no RBS were also established. We were also able to provide good design strategies for CAs, NCAs, and split-JOIN structures in them. With these, we would be able to facilitate the formalism of the other notions of soundness and their relationships, among others, for RDLTs. RDLTs that represent complex real-world systems can now be verified for classical soundness.

As a final note in this paper, we were able to pose conjectures (Appendix III) for future researchers to extend our results to RDLTs with RBS while recognizing possible reuse of level-based analysis and verification of classical soundness for them.

NOTES ON APPENDICES

The complete appendices section of the study is accessible at https://philjournsci.dost.gov.ph

REFERENCES

CORMEN T, LEISERSON C, RIVEST R, STEIN C. 2009. Introduction to Algorithms 3rd Edition. The MIT Press.

DELOS REYES R, AGNES K, MALINAO J, JUAYONG RAR. 2018. Matrix Representation and Automation of Verification of Soundness of Robustness Diagram with Loop and Time Controls. Proceedings of the Workshop on Computation, Theory, and Practice (WCTP).

JOHNSON DB. 1975. Finding all the Elementary Circuits of a Directed Graph. SIAM J. Comp 4: 77–84

LEOPOLD H, MENDLING J, GUNTHER O. 2015. What we can learn from quality issues of BPMN models from industry. IEEE Software 33(4): 1–9.

LOPEZ JCL, BAYUGA MJ, JUAYONG RA, MALINAO J, CARO J, TEE M. 2020. Workflow models for integrated disease surveillance and response systems. In: Theory and Practice of Computation. London: Taylor and Francis Group.

MALINAO J. 2017. On building multidimensional workflow models for complex systems modelling [Dissertation]. TU Wien, Austria. https://doi.org/10.34726/hss.2017.43523

MALINAO J, JUDEX F, SELKE T, ZUCKER G, ADORNA H, CARO J, KROPATSCH W. 2016. Robustness Diagram with Loop and Time Controls for System Modelling and Scenario Extraction with Energy System Applications. Energy Procedia 88: 537–543.

MALINAO J, TIU K, LOZANO LM, PASCUA S, CHUA RB, MAGBOO MS, CARO J. 2013. A Metric for User Requirements Traceability in Sequence, Class Diagrams, and Lines-of-code *via* Robustness Diagrams. Proceedings in Information and Communications Technology 7: 50–63.

QUIWA E. 2007. Data Structures. Alexan Publishing.

REZK A. 2012. A Theoretical and Experimental Investigation of Silica Gel/Water Adsorption Refrigeration Systems [Dissertation]. University of Birmingham.

STIEHL V. 2014. Process-driven applications with BPMN. Switzerland: Springer International Publishing. DOI: 10.1007/978-3-319-07218-0

SULLA CN, MALINAO J. 2023. Mapping of Robustness Diagram with Loop and Time Controls to Petri Net with Considerations on Soundness. Lecture Notes in Networks and Systems, Vol. 784. Cham, Switzerland: Springer.

VAN DER AALST WMP, VAN HEE KM, TER HOFSTEDE AHM, SIDOROVA N, VERBEEK HMW, VOORHOEVE M, WYNN MT. 2011. Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects of Computing 23(3): 333–363.

VAN DER AALST WMP. 1996. Structural Characterizations of Sound Workflow Nets. In: Computing Science Reports 96/23. Einhoven University of Technology.

WESKE M. 2007. Business Process Management: Concepts, Languages, Architectures. Berlin-Heidelberg: Springer-Verlag.

YIU A, GARCIA J, MALINAO J, JUAYONG RA. 2018. On Model Decomposition of Multidimensional Workflow Diagrams. Proceedings of the WCTP.

APPENDIX I. STRUCTURAL AND BEHAVIORAL PROFILES OF RDLTS

Theorem 1. An RDLT R with a single source $s \in V$ and single sink vertex $o \in V$ is classical sound if R is L-safe.

Proof. Let R be an L-safe RDLT.

Assuming that R is not a classical sound. If R is not classical sound, then we can claim that:

1. There is an activity profile $S = \{S(1), S(2),\ldots, S(n)\}$, $(q,o) \in S(n)$, $q \in V$, $n \in \mathbb{N}$, in R, where $\exists(p, u) \in S[i]$, $1 \leq i < n$, such that $\nexists(u, y) \in \cup_{j=i+1}^{n} S(j)$. This means that S does not show proper termination through $(p, u) \in S[i]$, *i.e.* the child $y \in V$ of u is a pending task of an unfinished process in S.
2. R is not live. That is, there is some $(u, y) \in E$ that is not used by at least one activity profile in R.

For the first claim, (p, u) here can be either of the following:

1. (p, u) leads to an AND-join merging at $y \in V$, where $\exists(v, y) \in E$, where $C(u, y) \in \Sigma$, $C(u, y) \neq C(v, y)$, and $C(v, y)$ was not checked by the algorithm, thereby causing the non-resolution of the AND-JOIN at y. With this, both (u, y) and (v, y) were never found to be unconstrained/traversed by the algorithm, such that $\exists j$, $i < j \leq n$, where (u, y), $(v, y) \in S(j)$. Since R is L-safe, we know that the algorithm can only backtrack to the common ancestor x of u and v, *i.e.* split origin x, of this split-AND-JOIN structure. Furthermore, since R is L-safe, there is a path $P = x_1x_2\ldots x_m$, $m \in \mathbb{N}$, where $x_1 = x$, $x_{m-1} = v$, and $x_m = y$. By the nature of the algorithm, this path is explored by the algorithm because there are no other (unrelated) processes that lead out of x apart from those that eventually go to y. In the subsequent exploration of the algorithm on P, we know that it never backtracks nor goes out of this structure because of the following:
   [a] There is no critical arc whose maximum allowable number of traversals may be reached by the algorithm, rendering it unusable in the next steps of activity extraction.
   [b] There are no splits at any x_i, $i = 1, \ldots, m-1$, that can cause the algorithm to never reach y.
   [c] There are no JOINs merging at x_i, $i = 2, \ldots, m$ because this path has no process interruptions.
   [d] There are no condition duplicates to $C(v, y)$ ($C(u, y)$) and $L(v, y) = L(u, y)$; hence, there is no exploration of the algorithm that checks $C(u, y)$ ($C(v, y)$) but is unable to check $C(v, y)$($C(u, y)$), resulting to the AND-join to never be resolved.
   [e] Every arc of P is loop-safe; therefore, its L-value always allows the checking (and eventual traversal) of the algorithm on the arc.

   Hence, in this case, we arrive at a contradiction.
2. (p, u) leads to a MIX-JOIN merging at $y \in V$, where $C(u, y) = \epsilon$, and $\exists(v, y) \in E$, where $C(v, y) \in \Sigma$, such that $C(v, y)$ was never checked by the algorithm to conclude that both (v, y) and (u, y) are unconstrained, thus including them in S. Note that $C(u, y) = \epsilon$ because this is the only case that can make y a pending task whenever one of conditions $c = C(v, y) \in \Sigma$ of this MIX-JOIN is not considered by the algorithm. If $C(u, y)$ had been equal to c, then (u, y) is concluded to be unconstrained, thus included in S. Now that this is established, we prove that $C(v, y)$ is always checked in this MIX-JOIN subsequent to the backtrack incurred by the algorithm upon evaluating $C(u, y) = \epsilon$.

   Since R is L-safe, we can establish that (v, y) is eventually reached by the algorithm in the same way we have shown in our proof above against the first claim, with the exception of (1.d); notwithstanding, (1.a) to (1.c) and (1.e) are already sufficient for us to know that u is reachable by the algorithm from x. (note that we allow MIX-JOINs to have duplicates of conditions at the merge point y). Note that whenever v is reached and $C(v, y)$ is evaluated, this arc is deemed by the algorithm to be unconstrained because all other conditions at the merging point are the same as $C(v, y)$. With this, the algorithm traverses (v, y) and (u, y) and includes them in S. Hence, for this case, we arrive at a contradiction.
3. (p, u) is an arc in R that leads to an OR-JOIN or a path P that leads to an output $o' \in V$, where $o' \neq o$, where o is the sink vertex in R. With this, $\exists(a, b) \in S(n)$ of the activity profile, where $b \in V \setminus \{o\}$. This violates proper termination because we expect that the last reachability configuration contains arcs that end up at the final output vertex o. Since R is L-safe and the algorithm is sequential in nature, we know that the algorithm continuously checks and traverses every arc in P without any backtracks incurred because of the following:
   - There is an absence of AND- and MIX-JOINs along P that can cause at least one backtracking of the algorithm, inducing a pending task at the point of backtrack.
   - Every non-critical arc of P is loop-safe and every critical arc is safe; hence, the algorithm is always able to check and traverse non-critical arc, as well as every critical arc or its escape arc (which is loop-safe as well).

With these, the algorithm is able to completely evaluate and execute P until its terminal node o′∈V, where ∄(o′,v) ∈ E, v ∈ V. However, we know that there is exactly one sink node o ∈ V of R. Hence, o′ = o. Hence, in this case, we arrive at a contradiction.

For the second claim, (u, y) ∈ E can never be included in any activity profile in R. This means that (u, y) and its RDLT R must be designed in a way that either of the following is true:

1. There is no path from a source $s \in V$ of R to $q(u)$.
2. (u, y) is involved in an MIX- or an AND-JOIN in R that cannot be resolved by the activity extraction algorithm.

Since we know that R is L-safe, it is loop-safe; hence, there is at least one path from $s \in I$ to $u(y)$, where $I \in V$ is the set of source vertices in R.

For the second specification, an AND-JOIN or a MIX-JOIN with a pair of arcs $\{(u, y), (v, y) \in E\}$, $C(u, y) \neq C(v, y)$ can never be resolved in any activity extraction by the algorithm if it has the following profile: $C(v, y) \in \Sigma$, and $v \in \Omega(y)$; thus, $(v, y) \in E$ is a looping arc of y, and $\nexists w \in \alpha(y)$, where $(w, y) \in E$, and $C(w, y) = C(v, y)$. This means that every path from every source vertex in R to v must pass through y first before reaching v, and no duplicate condition $C(w, y)$ can be evaluated as a substitute for $C(v, y)$ for (u, y) and (w, y) to be deemed unconstrained; therefore, it can be traversed and included in at least one activity profile.

However, since we know that R is L-safe, we know that every MIX- and AND-JOIN structure does not allow process interruptions in any of the components of the processes that are involved in the split-MIX/JOIN structure, *i.e.* $\nexists(v, y) \in E$ that is not a part of at least one of such processes in this structure. This means that every such for every $(v, y) \in E$, $v \in \alpha(y)$.

Hence, we arrive at a contradiction. ■

Theorem 2. The extended RDLT R' of a multi-sink R is classical sound if R' is L-safe.

Proof. The proof follows from Theorem 1. ■

Theorem 3. Let R'' be the maximal substructure of a sink vertex $o \in V$ and its set of source vertices $I \subset V$ of a RDLT R. Let R''' be the extended RDLT of R''.

If R''' is L-safe, then R''' is classical sound.

Proof. The proof follows from Theorem 1.

SOME NOTES ON SOUNDNESS FOR MULTI-OUTPUT RDLTS

Our discussions have so far been focused on proving single output (extended) RDLTs and their classical soundness with regard to the discoverable activities in them. We note that activity profiles are extracted by the algorithm using non-deterministic choices on exploring processes from an input source $i \in V$ and a target output vertex $o \in V$. The algorithm only terminates when o has been reached. If and when we are provided a multi-source and single-sink vertex in R, we can always build its extended RDLT, or a variation thereof, *e.g.* only attach the dummy source vertex to a set of required source vertices in R for an activity to reach the sink vertex and then verify this updated model for classical soundness with activities starting with this dummy source.

Meanwhile, for R is a multi-sink RDLT, we might take interest in generating some or all the activities for a set of source vertices and a specific target sink vertex o and/or verifying its classical soundness. For this set of RDLT instances, the non-deterministic nature of the algorithm in generating this set of activities targeting o can fail us. That is, this nature may extract an activity profile S that contains an edge ending at o at the final configuration of $S(n)$ while also containing having a reachability configuration $S(k)$, $2 \leq k \leq n$, where $\exists(v,o') \in S(k)$, where o' is one of the sink vertices of R, where $o' \neq o$. Note that though the process that ends with such o' is indeed completed without pending tasks thereafter, we can still objectively qualify this instance to fail the definition of proper termination of the activity. This is because there is no path that can be traced from $S(k)$ to $S(n)$ that connects o' to our target sink vertex o.

Nevertheless, a simple update to the structure of R can rectify the aforementioned improper termination for multi-sink RDLTs. That is, for every $x \in V$ where $(x, o) \in E$, and $\exists(x, o') \in E$, set $C(x, o) = c$, $c \in \Sigma$, and establish a dummy arc $(o', o) \in E$, where $C(o', o) = \epsilon$, and $L(o', o) \in \mathbb{N}$, *i.e.* a MIX-JOIN from x as a split origin merging at the intended target o. Hence, if and when the algorithm non-deterministically chooses the path toward o', it would seek to resolve the join by backtracking to a path that eventually ends at our target sink vertex o. Note that these modifications do not violate the existing satisfaction of L-safeness of the original R, whenever true while addressing the matter of proper termination. When interpreting activities of R representing real-world systems, you may ignore every dummy arc (o', o) from the final reachability configuration of its set of activities.

APPENDIX II. ASYMPTOTIC ANALYSIS OF CLASSICAL SOUNDNESS IN RDLTS

Lemma 1. The time and space complexity to verify the loop-safeness of R is $O(c|E|^4)$ and $O(c\,|E|)$, respectively, where c is the number of cycles in the graph/RDLT R.

Proof. Shown below are the requirements to determine loop-safeness in R, as well as the time and space complexity of accomplishing these requirements. Note that we take advantage of well-known problems, their algorithms, and their time/space complexity, if any, to accomplish these requirements.

1. **Problem 1:** determine the connectedness of R.
   - [a] **Details:** Loop-safeness requires R is a connected graph. That is, there is at least one path from at least one source vertex to every vertex $v \in V$ of R.
   - [b] **Well-known algorithm solving the problem:** Depth-first search or breadth-first search (Cormen *et al.* 2009; Quiwa 2007)
   - [c] **Time complexity:** $O(|V| + |E|)$
   - [d] **Space complexity:** $O(|V| + |E|)$
2. **Problem 2:** Generate all simple/elementary cycles (Johnson 1975) of a digraph.
   - [a] **Details:** We need to extract cycles of R to determine critical and non-critical arcs of E.
   - [b] **Well-known algorithm solving the problem:** Johnson's algorithm (Johnson 1975)
   - [c] **Time complexity:** $O(|V| + |E|)(c + 1)$, where c is the number of cycles in the graph/RDLT R (Johnson 1975), with the inclusion of the time complexity of generating strongly connected components of a directed graph/RDLT R, *i.e.* $O(|V||\mathrm{E}|)$.
   - [d] **Space complexity:** $\mathrm{O}(|V| + |E|)$ of the graph/RDLT R (Johnson 1975)
3. **Problem 3:** get the minimum value from a set of values
   - [a] **Details:** get the set of critical and non-critical arcs of E in R by identifying the minimum L-value for all the arcs involved in every cycle.
   - [b] **Well-known algorithm solving the problem:** linear search
   - [c] **Time complexity: $O(|E|)$:** if we do this per cycle, we have $O(c|E|)$, where c is the number of cycles in the graph/RDLT R
   - [d] **Space complexity:** $O(c\,|E|)$
4. **Problem 4:** get the minimum value from a set of values
   - [a] **Details:** get the set of arcs $minL\_CA(c_k)$ that has the minimal L-value among a set of critical arcs of R that is found per cycle c_k, accounting intersecting components of other cycles with c_k in R
   - [b] **Well-known algorithm solving the problem:** linear search
   - [c] **Time complexity $O(|E|)$:** if we do this per cycle, we have $O(c|E|)$, where c is the number of cycles in the graph/RDLT R
   - [d] **Space complexity:** $O(c\,|E|)$
5. **Problem 5:** get the union of sets, and then find its intersection with another set
   - [a] **Details:** compute the coefficient I for every $(u, v) \in$ in $minL\_CA(c_k)$, for every cycle c_k in R per non-critical arc $(x, y) \in E$. Note that $|minL\_CA(c_k)| \in O(|E|)$.
   - [b] **Algorithm:** we can assign a number for every unique arc found in the input sets. Then, we apply a sorting algorithm to this set of numbers corresponding to the set of arcs from the original input. Sorting takes $O(|E| \log|E|)$ time. From the ordered set of numbers, scan and eliminate duplicates among them (*i.e.* ($O(|E|)$ time). With this removal, we can then identify the union of the original input sets. To get the intersection of the resulting union and the last input set, we can employ a scan-and-match scheme for this, *i.e.* $(O|E|^2)$. Doing this computation per $(u, v) \in$ in $minL\_CA(c_k)$, we use $(O|E|^3)$ time. Furthermore, doing this computation for every cycle c_k, we use a total time of $O(c|E|^3)$, where c is the number of cycles in R. Lastly, do this computation for all non-critical arc of R, we accumulate $O(c|E|^4)$ time complexity.
   - [c] **Time complexity:** $O(c|E|^4)$
   - [d] **Space complexity:** $O(c\,|E|)$
6. **Problem 6:** get the sum of a set of values
   - [a] **Details:** for every non-critical arc $(x, y) \in$ E, get $RU(x, y)$ using the computed value of the coefficient I for every $(u, v) \in minL\_CA(c_k)$ of every cycle c_k in R;
   - [b] **Algorithm:** summation of a set of values

[c] **Time complexity:** we note that $|minL\_CA(c_k)| \in O(|E|)$. This is the number of addends in $RU(x, y)$ per non-critical arc $(x, y) \in E$. Thus, computing for $RU(x, y)$ takes $O(|E|)$ time. Doing this computation for all $(x, y) \in E$, we have a total time complexity of $O(|E|^2)$.

[d] **Space complexity:** $O(c\,|E|)$

The dominating problem from Problems 1–6 above in terms of computational complexity of solving it is Problem 5. Thus, the loop-safeness of R can be determined in $O(c|E|^4)$ time using $O(c\,|E|)$ space. ■

Lemma 2. Given an RDLT R that has a loop-safe set of non-critical arcs, the time and space complexity of verifying that R has safe critical arcs is $O(|E|^2)$ and $O(|V| + |E|)$, respectively.

Proof. To prove the safeness of every critical arc $(x, y) \in E$ of R, where there is a loop-safe set of non-critical arcs, we generate the set of escape arcs for (x, y). We do this for every such arc in E. For these computations, we obtain an $O(|E|^2)$ and $O(|V| + |E|)$ time and space complexity. Note that we already know which sets of arcs in R are critical and non-critical since this is part of our input, as well as the information on loop-safeness of every non-critical arc. Thus, checking for this information for the set of escape arcs we have generated per critical arc of each cycle is constant time. Therefore, to verify if all the critical arcs of R are safe takes $O(|E|^2)$ and $O(|V| + |E|)$ time and space complexity. ■

Lemma 3. Given an RDLT R that has a loop-safe set of non-critical arcs and safe critical arcs, the time and space complexity of verifying that R is JOIN-safe is $O(|V|\,|E|^2)$ and $O(|E|^2)$, respectively.

Proof. Shown below are the requirements to determine loop-safeness in R, as well as the time and space complexity of accomplishing these requirements.

1. **Problem 1:** determine if every MIX-JOIN and AND-JOIN at $y \in V$ has a one split origin at x.

   Algorithm:

   [b] Generate the set of vertices $V' \in V$, where $y \in V'$, whose in-degree is greater than 1 and is a merging point of either a MIX-JOIN, AND-JOIN, or OR-JOIN. This process takes $O(|E|^2)$ time and space complexity.

   [c] For every pair $(u, y), (v, y) \in E$, where $y \in V'$, find a path $P(q, y)$, $q \in \{u, y\}$, in R where $P(q, y) = p_{1;q}p_{2;q} \ldots p_{nq;q}$ such that $p_{nq-1;q} = q$ and $x_{nq;q} = y$, and $p_{1;q}$ has an out-degree greater than 1, and for every $p_{i;q}$, $2 \leq i \leq n_{q-1}$, $p_{i;q}$ has an out-degree of 1. Here, $p_{1;q}$ is a candidate split origin of the structure with respect to parent q of y.

   If there is such a pair $(u, y), (v, y) \in E$, where $p_{1;u} \neq p_{1;}v$, then the one split origin requirement is not satisfied for the processes $P(u, y)$ and $P(v, y)$, as well as for the entire MIX- or AND-JOIN merging at y. This process takes $(|E|^2)$ time and space complexity.

   Since we do this for every $y \in V'$, then this step takes $O(|V|\,|E|^2)$ and $O(|E|^2)$ time and space complexity, respectively.

2. **Problem 2:** determine if there are no unrelated processes at the split origin for every MIX- and AND-JOIN.

3. **Problem 3:** determine if at least one process in every MIX- and AND-JOIN structure branches out of it.

9. The results from [1] can be reused to solve this problem. That is, if there is one split origin for all processes of the MIX- or AND-JOIN structure that merges at y, then there is no branching that happens outside of the structure for each of these processes. Thus, this step also takes $O(|V|\,|E|^2)$ and $O(|E|^2)$ time and space complexity, respectively.

4. **Problem 4:** determine if there are no process interruptions for every process for every MIX- and AND-JOIN structure in R.

   Given the one split origin x from [1] of the structure merging at y through its parent $q \in V$, reuse $P(q, y)$ and check if every vertex $p_{i;q}$ in $P(q, y)$, except for y, is connected to some $s \in V$, *i.e.* $(s, p_{i;q}) \in E$ where $s \in V \setminus Lit(P(q, y))$. If this is true, then there is a process interruption at $P(q, y)$ of this structure; therefore, this requirement is not satisfied. This step takes $O(|E|^2)$ time and space complexity. Doing this evaluation for every MIX- and AND-JOIN structure in R yields $O(|V|\,|E|)$ and $O(|E|^2)$ time and space complexity, respectively.

5. **Problem 5:** determine duplicate conditions at the merge point y. This problem can be solved by sorting the conditions first, and thereafter, performing a linear-time scan through adjacent items in the sorted list of conditions to determine (non)duplicates. Doing this for every $y \in V'$, we get $O(|V||E| \log |E|)$ and $(O(|E|^2)$ time and space complexity, respectively.

6. **Problem 6:** determine the equality of the L-values at the AND-JOIN. A linear scan of the L-values of the applicable arcs of the AND-JOIN can solve this problem. Therefore, doing this for every $y \in V'$ takes $O(|V|\,|E|)$ time and space complexity.

7. **Problem 7:** determine the loop-safeness of the components of the JOIN. This is accomplished at a constant time since we are already given the input to

have loop-safe components.

Lastly, for any $y \in V'$ in [1], we know that we only need to check for loop-safeness of all its non-critical arcs and the safeness of every critical arc. The time and space complexity to generate the results for this problem follows from Lemma 2. Note too that the input to this step also provides that every non-critical arc is loop-safe and every critical arc is safe.

Therefore, given the input specifications of R, its JOIN-safeness can be determined with $O(|V||E|^2)$ and $O(|E|^2)$ time and space complexity, respectively. ■

Theorem 4. Verifying that an RDLT R is L-safe takes $O(c|E|^4)$ and $O(|E|^2)$ time and space complexity.

Proof. Follows from Lemmas 1, 2, and 3. ■

Corollary 1. Verifying that an RDLT R is classical sound takes $O(c|E|^4)$ and $O(|E|^2)$ time and space complexity.

Proof. Follows from Theorem 4.

APPENDIX III. SOME NOTES ON RESET-BOUND SUBSYSTEMS (RBSS)

Model verification of an RDLT R with at least one RBS relies on analyzing the model by constructing levels of abstractions *via* vertex simplification, as shown in Appendix Figure I. This produces Level 1 R_1 and Level 2 R_2 simplifications of R, wherein Level 1 focuses on the structures outside of an RBS, whereas Level 2 on the RBS itself. Both R_1 and R_2 become a multigraph, all vertices become controllers, and L-values are dropped. Furthermore, C-values are either preserved or transformed into ϵ based on type-alikeness of arcs going towards or out from a set of vertices in R_1 and R_2. R_1 only preserves a vertex from an RBS of R if it interacts with the environment of RBS *via* its in- and out-bridges, *e.g.* x_3, x_6, and x_7. The in-bridge of x_7 is (x_8, x_7), whereas its out-bridge is (x_7, x_9).

There are at most two sets of type-alike arcs for every vertex x – one set includes those that are in- or out-bridges of x, and the other are those that are neither in- nor out-bridges of x. For example, the type alike-arcs of x_7 are the sets $\{(x_8, x_7), (x_7, x_9)\}$ and $\{(x_3, x_7), (x_5, x_7), (x_6, x_7)\}$. Only the C-values of type-alike arcs of x are used by the algorithm in determining whether any arc ending at x is unconstrained and can be traversed. For example, the algorithm only requires that it is able to check the type-alike arcs (x_5, x_7) and (x_6, x_7) and evaluate C-values, *i.e.* $C(x_5, x_7) = b$ and $C(x_6, x_7) = a$ to determine if x_7 is reachable. That is, it does not need to check and evaluate $C(x_8, x_7)$ to reach x_7 using the process flows inside the RBS of R. The converse is also true if our algorithm first evaluates $C(x_8,$

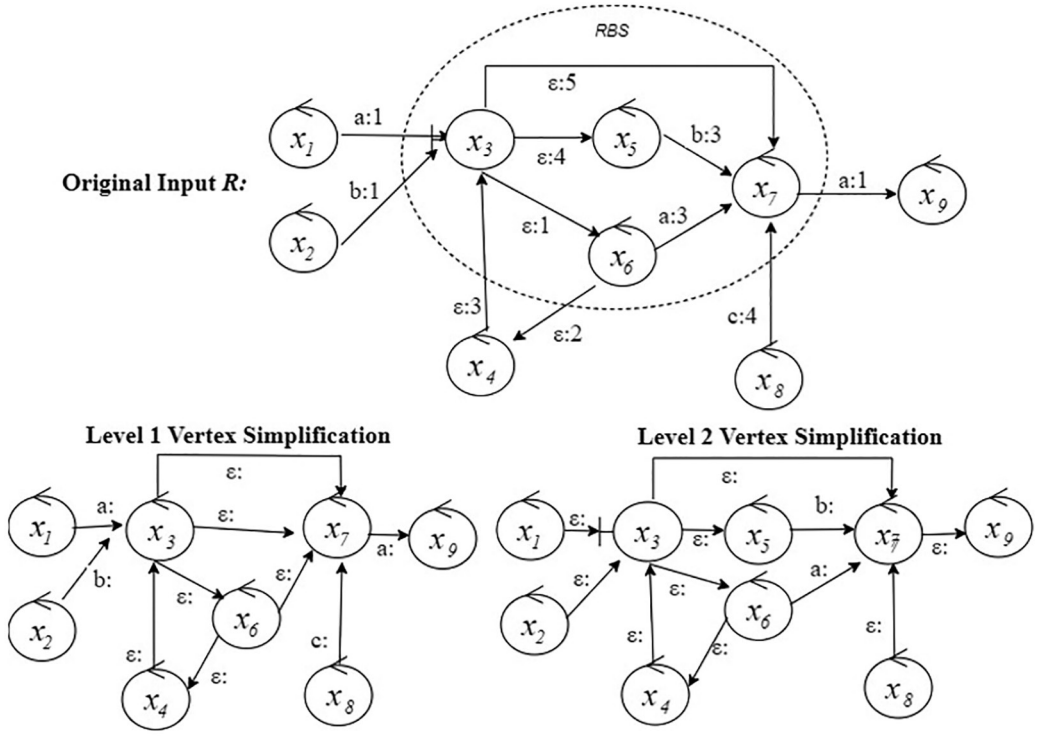

Figure I. Decomposing an input RDLT R by vertex simplification to obtain its Level 1 and Level 2 simplification.

x_9) = c to check and then immediately traverse (x_8, x_7) without the need to satisfy $C(x_5, x_7) = b$ and $C(x_6, x_7) = a$. This type-alikeness-driven evaluation by the algorithm is the foundation why we ignore the internal structures of an RBS in R – simply representing paths in these structures as one arc with ϵ C-values, in Level 1, and conversely, we also use ϵ C-values for the arcs coming from the outside of the RBS in Level 2. For the latter, we focus on analyzing the internal structure of the RBS itself [the details and formalism of vertex simplification can be found in the work of Malinao (2017)].

Obtaining R_1 and R_2 has indeed been helpful in verifying some properties in RDLTs, *e.g.* relaxed soundness. The usual approach in literature (Malinao 2017) is to verify the target RDLT property separately for R_1 and R_2 and use it, if applicable, to generalize the verification on the original input R. In a similar fashion, we can use the concepts and strategies of this study to verify classical soundness for the original R using R_1 and R_2. This can only be pursued if we first establish a strategy to represent the L-values of R in R_1 and R_2 loses the L-values, in an analogous manner as in their C-values. Nevertheless, future work on this can build from the following initial conjectures:

1. **Critical arcs should not be present in every RBS.** The algorithm resets the T-values of the internal arcs of an RBS whenever it traverses an out-bridge therein. We have already established that the L-values of every critical arc are predetermined and fixed all throughout any activity extraction. The resets "forget" the traversals that had been made on these internal arcs; thus, the usual checking as to whether or not the maximum number of traversals on them, as noted in their L-values, is not achieved. That is, the reset marks the arcs as never been traversed at all by the algorithm.

Meanwhile, we have also established that L-values of critical arcs should be the basis of the goodness of the L-values of some non-critical arcs of R. Therefore, putting critical arcs as internal arcs of an RBS establishes their dependency on the L-values of a set of in and out-bridges of some vertices of the RBS. This is because the maximum number of traversals as per the L-value of a critical arc inside an RBS gets extended as resets happen when the algorithm goes in and out of this RBS through these bridges.

Appendix Figure I shows the critical arc (x_3, x_6) of the cycle [$x_3x_6x_4x_3$] in R. Note that $L(x_3, x_6) = 1$ – if (x_3, x_6) is not inside an RBS, the algorithm can traverse this arc at most once. However, the reset of the arc's T-values, upon the algorithm's traversal of the out-bridge (x_6, x_4) and its subsequent re-entry to the RBS *via* x_3 of this cycle, will mean that (x_3, x_2) can be traversed at most two times – with $L(x_6, x_4)$ that seemingly acts as the critical value of this cycle, rather than (x_3, x_2).

2. **The L-value of an arc in R_2 representing a path in R is at least the minimum L-value along this path.** A path from x to y in R can be simplified as a single arc (x, y) in R_1 by vertex simplification. This simplification is founded on ignoring the internal vertices of this path due to model abstraction at Level 1. It assumes the guaranteed reachability of y from x in terms of C-value evaluation along such a path; thus, $C(x, y) = \epsilon$ in R_1 (the evaluation of this path happens in R_2). Using the same foundation, a good value for $L(x, y)$ in R_2 should be at least the minimum L-value among the L-values along this path in R_2.

Similar to determining the reusability of an arc, the most restrictive L-value along a series of arcs would be the minimum L-value among them. For example, a good L-value for the path $x_3x_5x_7$ of R that is abstracted as (x_3, x_7) in R_1 is $L(x_3, x_7) > (L(x_5, x_7) = 3)$. The final choice for $L(x, y)$, where (x, y) is the representative arc in R_1 of a path in R should also consider disjoint and intersecting cycles, as well as AND-JOINs safeness, that are present along these paths.

3. **R_1 and R_2 as vertex-simplified multi-digraph preserves L-safeness of R (whenever true).** The loss of some internal nodes of R in its corresponding R_1 and R_2 does not mean the loss of process paths in the latter. For example, a split at x leading to a JOIN at y using k-processes in R will still be represented in R_1, hence, the requirements for L-safeness are still verifiable in R_1 as in R.

Appendix Figure I shows how process flows are still preserved in the vertex simplification of R in R_1 and R_2. Note that there is a process branching for the AND-JOIN in R at (x_6, x_4). This branching is reflected both in R_1 and R_2, where R_1 consistently shows the process flows with abstractions on paths and C-values of the RBS while R_2 details them. Note that R is not L-safe and so are R_1 and R_2 due to the unsatisfiable condition of JOIN-safe structures for this property.

4. **L-safeness verification can be done in R_1 and R_2 and then generalized for the original R, provided that no critical arcs are found in every RBS of R.**