Matrix Representation of the Verification of Weak and Easy Soundness of Robustness Diagram with Loops and Time Contol

Nathaniel Enrique C. Eulin

April 18, 2025

Contents

Chapter 1

Introduction

1.1 Background of the Study

A workflow consists of three dimensions, namely the process, resource, and case dimension [1]. The process dimension is a specification of a process, the process being a partial ordering of a set of tasks such as control schemes like sequential, conditional, or iteration. The resource dimension pertains to the resource specification, where a resource is an object in the system that performs calculations, or processes, a task. Finally, the case dimension is the specification of the case, where case refers to the abstraction of a set of entities executed according to how the process is defined by the proper resource. [2] [6]. Workflow models use complex and dynamic real-world systems in various fields such as human resource management, biology (integrated disease surveillance [Lopez et al. 2020]), and manufacturing engineering (chiller systems [11]). This includes models such as Petri nets (PN) and Robustness Diagrams with Loop and Time Controls (RDLT) [6]. These models allow workflow components to be analyzed and model properties to be verified.

The study of the workflow model RDLT has been important in recent years due to its capability to represent a workflow in all of its dimensions (process, resource, and case). Although there are other workflow models that can represent all dimensions e.g. busi-

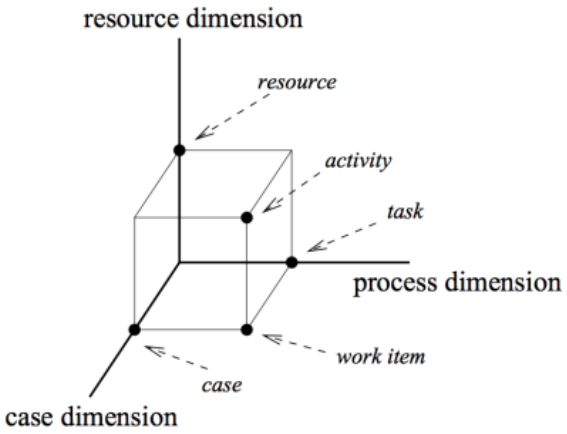


Figure 1.1: The three dimensions of workflows. (Image source:[6])

ness process modeling and notation (BPMN), activity diagrams, etc., there are gaps and challenges in verifying the properties of these models. Issues like concept excess, lack of support for explicit representation of data and rules, functional decomposition, etc. This is partly due to the separation of the represenation of the resources in a workflow to the actual processes. RDLT solves such problems, for example, by using abstractions of encapsulated activities that can be controlled by external processes (e.g., Reset-Bound Systems). In short, RDLT provides mechanims that effectively intgerate all three workflow dimensions. [6]

Soundness in Workflow Nets, eventually refered to as classical soundness, refers to the property of a workflow to be absent of livelocks, deadlocks, and other anomalies in the flow [2]. In general, to be classically sound, a model should have proper termination and liveness. There are other variations of soundness focusing on specific relaxations or modifications of the conditions proper termination and liveness, namely, relaxed, weak, easy, and lazy soundness. In the context of RDLTs, being a relatively newer workflow model, current literature shows some formalizations on notions of soundness. Classical soundness in RDLT was defined in [7] as well as relaxed soundness. Weak and easy soundness was later defined and formalized as well in [11]. Meanwhile, easy soundess in the context of RDLT is yet to be properly defined.

Hence, with all the knowledge about workflows and RDLT's notions of soundness, further research would expand in terms of automation to verify model properties. Therefore, various research has been done about the representation of RDLT into mathematical models aiming to automize verification. In Karen and Roben's work [5], they proposed a matrix representation of RDLT to verify soundness. Asoy [4] used a matrix representation of RDLT to verify and automate classical soundness. This paper aims to leverage the structural and behavioral profile of weak and easy soundness as formally defined in [11] and use matrix representations and operations to verify these notions of soundness in an RDLT.

The content of this paper is outlined as follows: Section 1.2 describes the basic definitions and notations of the current literature necessary to understand the topic at hand. Under this section describes brief background on Workflow Nets, Petri Nets, and its notions of soundness. It follows then the discussion on formal definitions of RDLT and its notions of soundness. Important algorithms in RDLT such as activity extraction (algorithm **??**) and vertex simplifications (algorithms 1.8 **??**) are described as well, including the concepts and definitions necessary. Section 1.3 to 1.7 outlines this paper's problem statement, aim, scope and limitations, significance, and the theoretical and conceptual framework of the study, respectively. Chapter 2 describes the methodology and important findings of important literatures.

1.2 Basic Definitions and Notations

Petri Nets

Petri Net [10] is a type of workflow model conceptualized by Carl Adam Petri in his disseration *Kommunikation mit Automaten* (Communication with Automata) in 1962. This model became the basis for most modelling framework, or was inspired thereof [6].

Although having to only represent a workflow of its case and process dimensions, understanding Petri Nets and its notions of soundness (as in Workflow Nets) can also aid in our understanding of the notions of soundness in RDLT, as it is what these prior notions of soundness were built upon.

A Petri Net is a bipartite graph composed of two types of nodes, namely, places (represented by a circle) and transitions (represented by a rectangle), and an arc, which connects nodes only of different types (place to transtion, and vice versa).
A place can either be an *input* or *output* place. A place is said to be satisfied if there is at least one token inside of the place, and if it is satisfied (also called enabled), it may fire in which every output place it has recieves one token [10].

The following sections defines Petri Nets and Worklow Nets, which is a particular class of Petri Net, and follows the definitions of notions of soundness in Workflow Nets. These are important to understand as they are the basis for the formulization of the notions of soundness in the context of RDLT. The definitions in this section are mainly taken from papers [6] [7] [3] [2]with descriptions for supplemantary explanations.

Definition 1.2.1. Petri Nets and Markings [2] [7] A Petri Net is (P, T, F, M_0) which is a tuple that consists of four elements:

- a set of places P
- a set of transition T where the intersection of places and transitions must not be empty
- a set of arcs F which is a subset of the union of all the possible connections from places to transitions and vice versa.
- an initial marking $M_0 : P \to \mathbb{N}$

Figure 1.2 shows the components of a Petri Net.

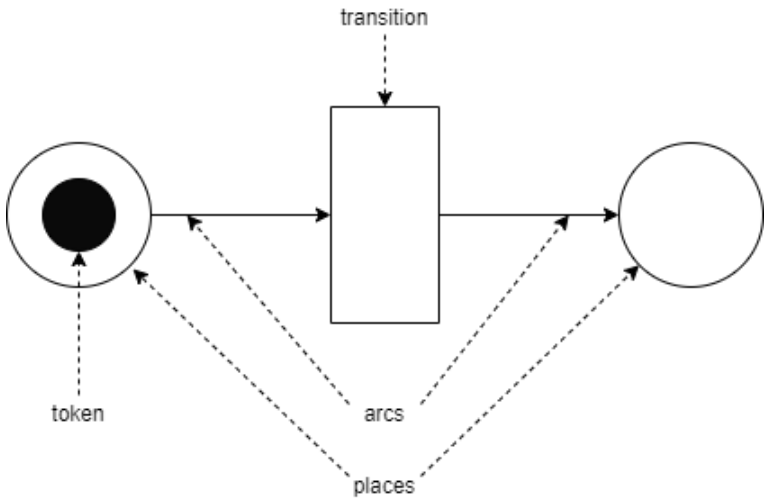


Figure 1.2: The components of a Petri Net

Soundness in Workflows

A *Workflow Net* (WF-Net) is a class of Petri Net that offer capabilities of the analysis of parallel execution of activities across splits and joins of processes [6].

Definition 1.2.2. Strongly Connected [6]
A Petri Net is **strongly connected** if there exists a path from x to y for every pair of nodes x and y.

Figure 1.2 shows a simple Petri Net where every possible pair of nodes are connected, hence a **strongly connected** Petri Net.

Definition 1.2.3. Workflow Nets [6]
A Petri Net is a **Workflow Net** if such a Petri Net holds the following conditions:

- The Petri Net contains two special places which are a source place i that does not have any input places and a sink place o that does not have any output places.
- The Petri Net is strongly connected if the source place i and the sink place o are connected to a transition $t*$, as an input and output place respectively.

Given the two conditions outlined above, this would mean that the Petri Net in Figure 1.2 is a Workflow Net.

On the literature of Workflow Nets, all notions of soundness are already established. The notions of soundness in the context of Workflow Nets are shown in a heirarchy in Figure 1.3, where arrows show a relationship of implication.

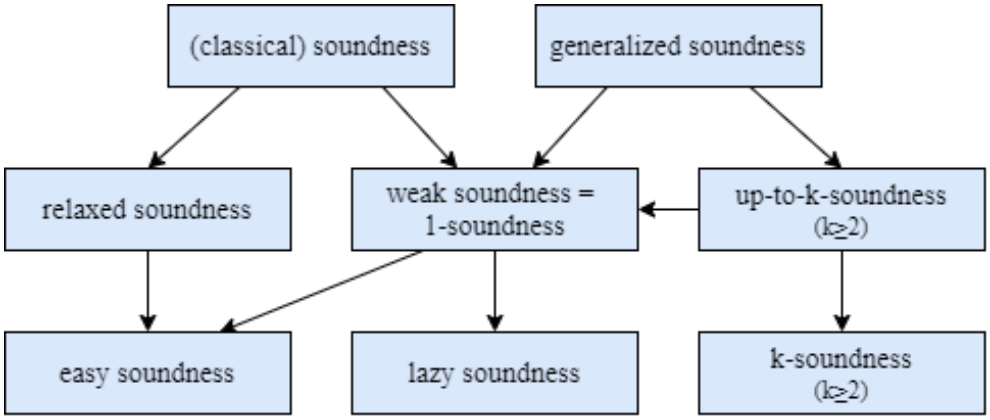


Figure 1.3: Relationships between the different soundness notions where the arrows represent the implication of relationships from one notion to another. (Image source: [3])

In the next sections, only classical soundness, relaxed soundness, weak, and easy soundness are discussed, as they are the ones relevant to understanding weak and easy soundness in the context of RDLT, to be discussed in further sections.

Definition 1.2.4. Classical Soundness of Workflow Nets [3]
A Workflow Net is classical sound if and only if the following requirements are satisfied:

- Option to complete: the sink place o is reachable from the source place i.
- Proper termination: the sink place o receives exactly one token and all other places must be empty.

- Liveness: every transition must be involved in at least one process specification.

This notion of soundness offeres a very strict criterion since it imposes that all process for each case are complete, leaving no pending, or cancelled task when the token reaches the sink, as explained in condition 2. [6] Figure 1.2 shows a classically sound workflow net. Sytems also allow the implementation of a workflow wherein at least every transition is involved in a case that is properly completed, or a token reaches the sink. This variation of soundness is less restrictive compared to that of a classically sound workflow net. The criterion is weakened to some degree, hence the following notions of soundness:

Definition 1.2.5. Relaxed Soundness of Workflow Nets [3]
A Workflow Net is of relaxed sound if and only if the following requirements are satisfied:

- Option to complete: the sink place o is reachable from the source place i.
- Proper termination (weakened): at least one case exists that completes with one token in the sink place o with other places being empty.
- Liveness: every transition must be involved in at least one process specification.

In relaxed soundness, the proper termination is weakened to at least one case in the workflow net exist where it completes with one token in the sink o while other places are empty. Therefore, it allows other cases where the workflow terminates while still having processes working in the background, represented in the model by a token inside a place, as shown in Figure 1.4. Similarly, since classical soundness is implied in a relaxed sound workflow net, Figure 1.2 is also of relaxed sound.

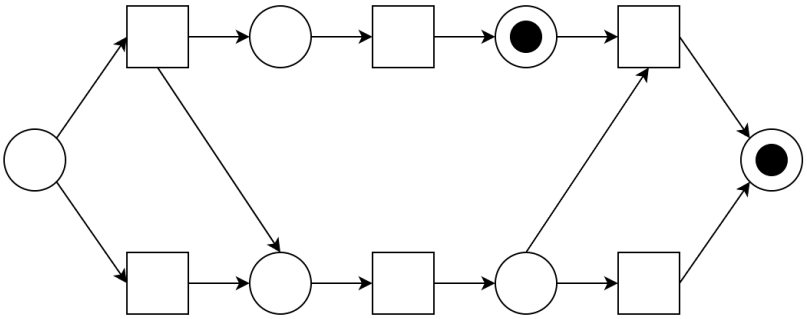


Figure 1.4: A Workflow Net of Relaxed Soundness (Adopted From: [3])

Definition 1.2.6. Weak Soundness of Workflow Nets [6]
A Workflow Net is of weak sound if and only if the following requirements are satisfied:

- Option to complete: the sink place o is reachable from the source place i.
- Proper termination: the sink place o receives exactly one token and all other places must be empty.

On weak soundness in workflow nets, only two conditions are required, as defined above. The workflow should have an option to complete, and it should properly terminate. In comparison to classical soundness, it does not require liveness, that is every transition be live for every execution. An example of a weak sound workflow net is found in Figure

1.5. This net is not classically sound since t_3 is a dead transition, making the workflow net not live and not fulfilling the third condition of a classically sound workflow net.

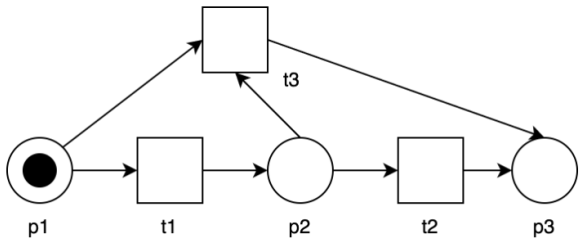


Figure 1.5: A Workflow Net of Weak Soundness (Adopted from: [3])

Definition 1.2.7. Easy Soundness of Workflow Nets [6]
A Workflow Net is of easy sound if and only if the following requirement is satisfied:

- Option to complete: the sink place o is reachable from the source place i.

Easy soundness in the context of workflow nets only has one condition as defined above, that is it should have an option to complete. Similarly, in Figure 1.6, transition t_1 to t_2 leads to the sink o, hence has an option to complete. However, it does not satisfy the other conditions of other notions of soundness.

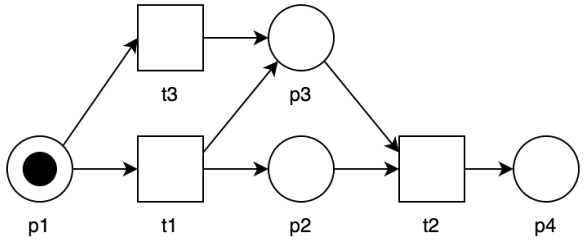


Figure 1.6: A Workflow Net of Easy Soundness (Image source: [3])

As a summary and ease of reference, the table below maps the requirements of soundness in the context of workflow nets **??**, obtained from [11].

Robustness Diagram with Loop and Time Controls

In this section, RDLT and its notions of soundness are defined. RLDT is a workflow model that allows all three dimension of a workflow to be represented.

Definition 1.2.8. RDLT [6]
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 each vertex has a type $V_{type} : V \rightarrow$ {'b', 'e', 'c'} where 'b', 'e', and 'c' means the vertex is either a "boundary object", an "entity object", or a "controller", respectively.

- A finite set of arcs $E \subseteq (V \times V) \backslash E'$ where $E' = \{(x, y) | x, y \in V, V_{type}(x) \in \{$'b', 'e'$\}$, $V_{type}(y) \in \{$'b', 'e'$\}\}$ with the following attributes with user-defined values,

  - $C : E \to \Sigma \cup \{\epsilon\}$ where Σ is a finite non-empty set of symbols and ϵ is the empty string. Note that for real-world systems, a task $v \in V$, i.e. $V_{type}(v) =$'c', is executed by a component $u \in V, V_{type}(u) \in \{$'b','e'$\}$. This component-task association is represented by the arc $(u, v) \in E$ where $C((u, v)) = \epsilon$. Furthermore, $C((x, y)) \in \Sigma$ represents a constraint to be satisfied to reach y from x. This constraint can represent either an input requirement or a parameter $C((x, y))$ which needs to be satisfied to proceed from using the component/task x to y. $C((x, y)) = \epsilon$ represents a constraint-free process flow to reach y from x or a self-loop when $x = y$.
  - $L : E \to \mathbb{Z}$ is the maximum number of traversals allowed on the arc.

- Let T be a mapping such that $T((x, y)) = (t1, ..., tn)$ 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.

- $M : V \to \{0, 1\}$ indicates whether $u \in V$ and every $v \in V$ where $(u, v) \in E$ and $C((u, v)) = \epsilon$ induce a sub-graph G_u of R known as a **reset-bound subsystem** (RBS). The RBS G_u is induced with the said vertices when $M(u) = 1$. In this case, u is referred to as the **center** of the RBS G_u. G_u's vertex set V_{G_u} contains u and every such v, and its arc set E_{G_u} has $(x, y) \in E$ if $x, y \in V_{G_u}$.

  Finally, $(a, b) \in E$ is called an **in-bridge** of b if $a \notin V_{G_u}, b \in V_{G_u}$. Meanwhile, $(b, a) \in E$ is called an **out-bridge** of b if $b \in V_{G_u}$ and $a \notin V_{G_u}$. Arcs $(a, b), (c, d) \in E$ are **type-alike** if $\exists y \in V$ where $(a, b), (c, d) \in Bridges(y)$ with $Bridges(y) = \{(r, s) \in E | (r, s)$ is either an in-bridge or out-bridge of $y\}$ or if $\forall y \in V, (a, b), (c, d) \notin Bridges(y)$.

Figure 1.7 shows an RLDT with a reset-bound system with center x_2 and its owned controllers x_3 and x_4. The in-bridges of the RBS with respect to x_2 are (x_1, x_2) and (x_6, x_2). Both arcs are type-alike with respect to x_2 as defined in 1.2.8. However, (x_3, x_2), being inside the RBS, is not type-alike to any of the aforementioned arcs. (x_4, x_5) and (x_4, x_6) are type-alike with respect to x_4. For each arcs of this RDLT, $C-$ and $L-$ attributes are defined. The x_1 serves as the source, while the x_7 serves as the sink.

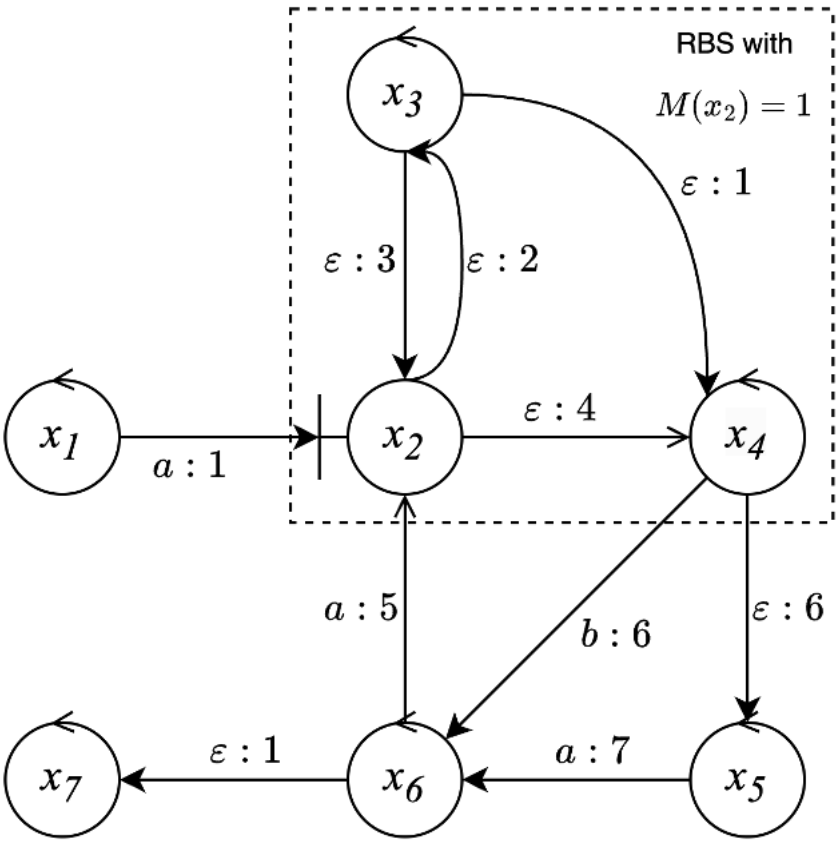


Figure 1.7: A Reset-Bound Subsystem-Containing Robustness Diagram with Loop and Time Controls with center at x_2, where $M(x_2) = 1$. (Adopted from: [8])

Activity Extraction Algorithm

The proposed Algorithm **??** in paper [6], called *Activity Extraction Algorithm*, takes an input RDLT R with a defined start index s and goal vertex f. As shown in Algorithm **??** it returns an *activity profile* defined in Definition 1.2.10, a set of exected vertices from the extracted activities. To understand Algorithm **??** fully, along with the definition of **activity profile**, **reachability configuration**, defined in Definition 1.2.9, is introduced. Algorithm **??** is found at the appendices of this paper.

To perform **activity extraction** from vertex x, where $x = s$ at the start of algorithm **??**, the algorithm performs a **check**, defined in Algorithm **??** in the appendices, on x and other vertex $w \in E$ where $(x, w) \in E$. A **check**, defined in algorithm **??** at the appendices, is a subroutine in $\mathcal{A}$ where $(x, y) \in E$ is arbitrarily selected, given that the number of traversals done on (x, y) has not yet reached $L((x, y))$, which is the traversal limit of the arc, if such y exists in V. Similarly, the constraint $C((x, y))$ is checked as well, and given both $L((x, y))$ and $C((x, y))$ are satisfied, then the arc (x, y) is traversed and the next vertex in the workflow will be y. If this valid vertex y is reached, the $\mathcal{A}$ assigns time step traversal value t_i. If a previous arc or an incoming arc (u, x) prior to (x, y), the time step t_i is set to the maximum time step from all the time traversals $T((u, x))$ plus 1 from the all the incoming arcs before (x, y). Otherwise, the time traversal value is set to 1, indicating the first traversal of the arc. After the check, $\mathcal{A}$ evaluates whether every $(v, y) \in E$, $v \in V$, does not prevent the traversal on (x, y). When there are no constraints

to the traversal on (x, y), the said arc is said to be an **unconstrained arc**, as defined in 1.2.11. On the other hand, if (x, y) and (v, y) are not type-alike, (x, y) will not prevent traversal of (x, y), and (x, y) remains unconstrained with respect to all type-alike arcs with respect to the vertex y.

The following are the formal definitions of the concepts mentioned.

Definition 1.2.9. Reachability Configuration [6]
A **reachability configuration** $S(t)$ in $R = (V, E, \Sigma, C, L, M)$ contains the arcs traversed by $\mathcal{A}$ at time step $t \in \mathbb{N}$.

Definition 1.2.10. Activity Profile [6]
A set $S = \{S(1), S(2), \ldots, S(d)\}$ of reachability configurations, $d \in \mathbb{N}$, is an **activity profile** in $R = (V, E, \Sigma, C, L, M)$ where $\exists (u, v) \in S(1)$ and $(x, y) \in S(d)$ such that $\nexists w, z \in V$ where $(w, u), (y, z) \in E$.

Essentially, a **reachability configuration** is the set of arcs traversed by activity extraction at a given time step t_i, while the **activity profile** of an RDLT is the set of all reachability configurations from time step t_0 to t_d, where d is the step the workflow reached the sink.

Given these definitions, and following $\mathcal{A}$, the RDLT in Figure 1.7 will have the following **reachability configuration**:

$$
\begin{aligned}
S(1) &= \{(x_1, x_2)\}, \\
S(2) &= \{(x_2, x_4)\}, \\
S(3) &= \{(x_4, x_6), (x_4, x_5), (x_5, x_6)\}, \\
S(4) &= \{(x_6, x_2)\}.
\end{aligned}
$$

The **activity profile** therefore of Figure 1.7 is $S = \{S(1), S(2), S(3)\}$.

In step 4.3 of algorithm $\mathcal{A}$, an evaluation is performed to determine if $(x, y) \epsilon E$ is an unconstrained arc. The definition of an unconstrained arc is defined as follows:

Definition 1.2.11. Unconstrained Arc [6]
An arc $(x, y) \in E$ is **unconstrained** if $\forall (v, y) \in E$, where (x, y) and (v, y) are type-alike, any of the following traversal conditions hold,

1. $C((v, y)) \in \{\epsilon, C((x, y))\}$,
2. $|\{t_i \in T((x, y)) | t_i \geq 1\}| \leq |\{t_j \in T((v, y)) | t_j \geq 1\}| \leq L((v, y))$,
3. $C((v, y)) \in \Sigma, C((x, y)) = \epsilon \wedge T(v, y) \neq [0]$.

Note that (x, y) will remain unconstrained (if it is such) regardless of any other (v, y) where (x, y) and (v, y) are not type-alike.

Vertex Simplification

In RDLTs, subsystems of a workflow cause an increase in the complexity of its analysis. **Vertex simplification** aims to reduce this complexity through abstraction of the components of these subsystems. In this context, the RBS is subsystem being simplified in the process of vertex simplification, thus streamlining the structure of the diagram without losing essential information necessary for analysis.

Definition 1.2.12. Vertex Simplified R [6]
A vertex-simplified RDLT $G = (V', E', C')$ of $R = (V, E, T, M)$ (with arc attributes C and L) is a multigraph whose vertices $v \in$ V have $V_{type}(v) =' c'$ where G is derived from R such that the following holds:

1. $x \in V'$ if any of the following holds:
   - $x \in V$ and $x \notin V_{G_u}$ of an RBS G_u in R, or
   - there exists an in-bridge $(q, x) \in E$ of $x \in V \cap V_{G_u}, q \in V$ of R, or
   - there exists an out-bridge $(x, q) \in E$ of $x \in V \cap V_{G_u}, q \in V$ of R, or
2. $(x, y) \in E'$ with $C'((x, y)) = C((x, y))$ for $x, y \in V'$ if $(x, y) \in E$
3. $C((x, y)) = \varepsilon$ if $x, y \in V' \cap V_{G_u}$ and x is an ancestor of y in R and $(x, y) \notin E_{G_u}$

We refer to this simplification of R as **level-1 vertex simplification** of R with respect to every RBS G_u in R. A **level-2 vertex simplification** of R with respect to its RBS G_u is the level-1 vertex-simplification of G_u where G_u is treated as an RDLT where the value of the vertex attribute M of u is redefined to 0, i.e. $M(u) = 0$. With this, the verification of model properties (i.e. maximally composed and sound RDLTs) are separately done for the level-1 and level-2 vertex simplifications of R. However, this separation does not affect the validity of proving for any of these properties on the entire RDLT itself.

Figure 1.8 shows the vertex simplified form of the sample RDLT in Figure 1.7.

Definition 1.2.13. Contraction of Arcs in RDLTs [6]
Given an RDLT $R = (V, E, T, M)$ and its vertex-simplified RDLT $G = (V', E', C')$, a contraction of $(x, y) \in E'$ of G, x, $y \in V'$, $x \neq y$, results to a multidigraph $G* = (V*, E*, C*)$ such that:

1. $V* = V' \backslash \{x, y\} \cup \{x'\}$. Here, x' is a dummy vertex representing x and y.
2. $E* = \Big\{ E' \bigcup\limits_{\forall z \in V', \exists (z,v)_i \in E', v \in \{x,y\}} \Big\{ \bigcup\limits_{\forall i} \big\{ (z, x')_i \big\} \Big\} \bigcup\limits_{\forall z \in V', \exists (v,z)_j \in E', v \in \{x,y\}} \Big\{ \bigcup\limits_{\forall j} \big\{ (x', z)_j \big\} \Big\} \Big\} \backslash$
   $\Big\{ \bigcup\limits_{\forall q \in V', \exists (q,v)_i \in E', v \in \{x,y\}} \Big\{ \bigcup\limits_{\forall i} \big\{ (q, v)_i \big\} \Big\} \bigcup\limits_{\forall q \in V', \exists (v,q)_j \in E', v \in \{x,y\}} \Big\{ \bigcup\limits_{\forall j} \big\{ (v, q)_j \big\} \Big\} \Big\}$,
   where $z \notin \{x, y\}$, $(a, b)_i \in E'$ is the i^{th} arc from $a \in V'$ to $b \in V'$, $i = 1, 2, \ldots, n$, where n is the outdegree of a.
3. $C * ((z, x')_i) = \varepsilon$, $\forall z \in V'$ where $\exists (z, y)_i \in E'$, $i \geq 1$,
4. $C * ((x', z)_i) = C'((v, z)_i)$, $v \in \{x, y\}$, $\forall z \in V'$ where $(v, z)_i \in E'$. $i \geq 1$.

Contraction of arcs in RDLT involves the merging of two vertices into one *dummy* vertex while still preserving the properties of the arcs involved. x and y vertices are merged into a single vertex x' and the outgoing and ingoing arcs are adjusted along with the associated constraints This allows the simplification of the RDLT by reducing the amount of vertices to be analyzed.

Definition 1.2.14. Feasible Contraction [6]
A contraction of (x, y) is feasible if $\nexists(u, x) \in E'$ where $C((u, x)) \in \Sigma$, $u \in V'$ and

$$\bigcup_{\forall i}\{C'((x,y)_i)\} \cup \{\varepsilon\} \supseteq \bigcup_{\forall z \in V', \text{ where } \exists (z,y)_j \in E', j \geq 1} \{\bigcup_{\forall i}\{C'((z,y)_j)\}\},$$

where $(x, y)_i \in E'$ is the i^{th} arc from x to y, $i = 1, \ldots, n$, and n is the outdegree of x.

By this definition, the validity of contraction of vertices x and y is evaluated. It is ensured in this condition that these vertices are correct hence can be contracted into a single vertex without violating any constraints of the arcs. This ensures unerroneaous behavior with the simplified RDLT.

Definition 1.2.15. Contraction Path in RDLTs [6]
Given an RDLT $R = (V, E, T, M)$ and its vertex-simplified RDLT $G_1 = (V_1, E_1, C_1)$, a contraction path from x_1^1 to x_n in G_1 is a sequence $p = x_1^1 x_2 \ldots x_n$, $n \leq |V_1|$, where a contraction is feasible on $(x_1^{i-1}, x_i) \in E_{i-1}$ in G_{i-1} resulting to $G_i = (V_i, E_i, C_i)$ for $i = 2, 3, \ldots, n$, and $x_1^i \in V_i$ represents $x_1^{i-1} \in V_{i-1}$ and $x_i \in V_{i-1}$ whose arc (x_1^{i-1}, x_i) is contracted.

In simpler terms, the creation of a contraction path of a given vertex simplified RDLT G_1 or G_2, which can also be represented as R_1 and R_2 respectively, is the merging of a pair of vertices x and y connected by the arc (x, y) from the initial vertex through the contraction of arcs. Such a merging is only possible if the set of C-values of all arcs from x to y is a superset of the C-values of all arcs from other vertices in the subgraph towards y [8]. This continues until the subgraphs R_1 and R_2 are represented as one vertex, if possible. This method is also referred to as the *graph contraction strategy.*

Definition 1.2.16. Sibling Processes [12] Given a split at $x \in V$ with its processes P and P', where $P = x_1, x_2, \ldots, x_n$, $P' = y_1, y_2, \ldots, y_m$, P and P' are called sibling processes if $x_1 = y_1$, $x_n = y_m$, and P and P' have no arcs in common.

In simpler terms, the vertices in R present in a vertex-simplified RDLT G are those that do not belong in any RBS or those inside an RBS but has an *in-bridge* and/or an *out-bridge.* The vertex-simplifications R_1 and R_2 of the RDLT in Figure 1.7 is shown in Figure 1.8, respectively. R_1 captures a subgraph of R found outside of its RBS, including the vertices of the RBS that have at least 1 in-brigde or out-bridge. R_2 vertex-simplification, however, captures the RBS itself. The vertices that have in- or out-bridges in R are marked either as sources and/or sinks in R_2.

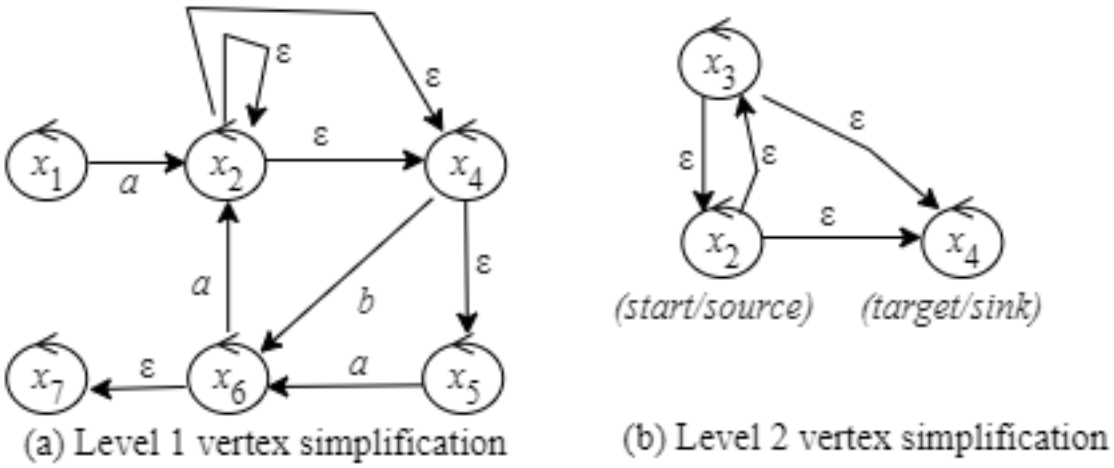


Figure 1.8: The levels 1 and 2 vertex simplification R_1 and R_2 of the RLDT in 1.7 (Image Source: [8])

L-Safeness of RDLTs

In the analysis of processes in RDLT, particularly on its verification of soundness property, it is crucial that possible L-attrbute-based deadlocks are avoided. Reachability of the vertices relies heavily on the configuration of the arcs' L-values, along with other factors such as connectivity, and C-attributes. The following definitions discusses the L-safeness with RDLTs from [7].

Definition 1.2.17. Critical and Escape Arcs [7] A cycle $c = [x_1 x_2 \dots x_n]$ is a sequence of vertices $x_i \in V$, where $(x_i, x_{i+1}) \in E$, $i = 1, 2, \dots, n-1$, 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 $v \in V \backslash Lit(c)$, 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.

An example of a **cycle** is x_2, x_3, where the arc from x_3 eventually leads to x_2, where x_3 can be traced, as well. The **critical arc** in this cycle is the arc with the minimum L-attribute, which is (x_2, x_3). Hence, the rest of the arcs in this cycle is a **non-critical arc**, e.g., the arc (x_3, xx_2). The **escape arc** will be the arc (x_2, x_4).

Definition 1.2.18. Loop-Safe Arcs [7] Let R be a connected RDLT, i.e. for every vertex $v \in V$, there is a path from a source vertex $s \in V$ to v in R. A non-critical arc (NCA) $(x, y) \in E$ of R is loop-safe if $L(x, y) > RU(x, y)$, where

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

$$I = \begin{cases} 1, & \text{if } k = 1 \text{ or } \cup_{j=1^{k-1}} minL\_CA(c_j) \cap minL\_CA(c_k) \\ 0, & \text{otherwise,} \end{cases}$$

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

Definition 1.2.19. Safe Critical Arcs [7] 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.

In general, escape arcs ensure that the workflow can escape from a cycle whenever the critical arc is fully used in an activity. However, having a safe critical arcs and loop-safe NCAs in R is insufficient in achieving complete soundess. In ensuring R is classically sound, an added requirement,that of $JOIN$-safe L-values, on an R having no RBS.

JOINs in RDLTs

In RDLTs, it is not impossible for multiple processes to converge at a single vertex (JOIN). The type of join specifies how the incoming processes interact before proceeding in the workflow. The following definitions are the different types of joins in RDLTs:

1. AND-JOIN at $y \in V$: For every type-alike arcs $(v, y), (u, y) \in E$ where $(v, y) \neq (u, y)$, their C-values $C(v, y) \neq C(u, y)$ and $C(v, y), C(u, y) \in \Sigma$.

2. MIX-JOIN at $y \in V$: There is at least one pair of type-alike arcs $(v, y), (u, y) \in E$ where $(v, y) \neq (u, y)$, $C(v, y) \neq C(u, y)$ and $C(v, y) = \epsilon$, and $C(u, y) \in \Sigma$.

3. OR-JOIN: For every type-alike arcs $(v, y), (u, y) \in E$ where $C(v, y) = C(u, y)$.

The concept of JOIN-safe L-values aids in ensuring the $JOIN$-safeness of an RDLT. This is one of the helpful design strategies to achieve classical soundness in JOINs, as discussed the paper [7]. This, along with defintions 1.2.18 and 1.2.19, provides the structural requirement for a classically sound RDLT, e.g, proper termination and liveness. These definitions serves as the basis for determining looser notions of soundness.

Definition 1.2.20. JOIN-safe L-values [7] 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 $P_u = a_1 a_2 \dots a_n$, where $a_1 = x$, $a_{n-1} = u$, $a_n = y$, $a_i \in V$, $1 < I < n - 2$, and another path $P_v = b_1 b_2 \dots b_m$, 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 $s \in V \backslash Lit(P_q)$.

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 $s \in V \backslash Lit(P_q)$.

5. **Duplicate conditions.**

- If $C(u, y)$, $C(v, y) \in \Sigma$, i.e. an AND-JOIN at y, then there is no process $P = [x_1 x_2 \dots x_p]$ in R, where $x_1 = x$, $x_p = y$, such that $C(x_{p-1}, x_p) = C(u, y)$ (or $C(v, y)$).
- Without any loss of generality, if $C(u, y) = \varepsilon$ and $C(v, y) \in \Sigma$, then any process $P = x_1 x_2 \dots x_p$ in R, where $x_1 = x$, $x_p = y$, can have $C(x_{p-1}, x_p) \in \{\varepsilon, C(v, y)\}$. 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 $P = x_1 x_2 \dots x_k$, $k \in \mathbb{N}$, where $x_1 = x$ and $x_k = y$, the arc $(x_1, x_{i+1}) \in E$. $i = 1, 2, \dots, k-1$, is loop-safe. Lastly, for an OR-JOIN merging at y, each process $P = x_1 x_2 \dots x_k$, $k \in \mathbb{N}$, of this JOIN has its arc (x_i, x_{i+1}) to have a JOIN-safe L-value, $i = 1, 2, \dots, k-1$, if (x_i, x_{i+1}) is either a loop-safe arc or a safe CA in R.

Reusability and Resets in RDLTs

The concept of reusability refers to how components of an RDLT can be used in different situations, such as when an RDLT has at least one RBS [8]. The information in this section is sourced from [8] unless explicitly mentioned otherwise.

Definition 1.2.21. Pseudocritical Arcs, Pseudo-escape Arcs
A pseudocritical arc $(PCA)(x, y) \in E$, where (x, y) is a component of a cycle c in R where $L(x, y)$ is the minimum among all the L-values of the arcs in c which are not arcs of an RBS in R.

Meanwhile, a pseudo-escape arc $(PEA)(x, z) \in E$ is a non-critical arc in R where (x, y) and (x, z) are type-alike.

Definition 1.2.22. Expanded Reusability in RDLTs with Resets
Let $R = (V, E, T, M)$ be a connected $RDLT$. If $\exists v \in V$ where $M(v) = 1$, then let $B = (V', E', T', M')$ be the RBS with its center v.

1. Let $IB_r(X)$ be the set of in-bridges of $x \in V$.

2. Let $Cycles_{part}(R)$ be a set of cycles in R with the following property: $\forall p = [x_1 x_2 ... x_n] \in Cycles_{part}(R)$, there exist cycle components meeting these conditions:

   (a) $(x_i, x_{i+1}) \in E$ is inside an RBS B of R (i.e, $x_i, x_{i+1} \in V'$).

   (b) $(x_j, x_{j+1}) \in E$ is not inside B (i.e, $x_j or x_{j+1} is not in V'$).

Whenever we have cycles $d = [d_1 d_2 ... d_{m1}], e = [e_1 e_2 ... e_{m1}] \in Cycles_{part}(R)$ that overlap in their non-RBS components, and this overlap contains both of their PCAs $(d_k, d_{k+1}), (e_{k'}, e_{k'+1})$, where $L(d_k, d_{k+1}) \geq L(e_{k'}, e_{k'+1})$, then consider only one of these PCAs. Choose the one with the minimum L-value as it would ultimately contribute to the reusability of every RBS component reachable from these cycle components.

Definition 1.2.23. (RU-preserving transformation)
A transformation δ of an input RDLT R to another RDLT R' is said to be RU-preserving if for every activity profile $S = S(1), S(2), ..., S(n_1), n_1 \in IN$, that is derivable from

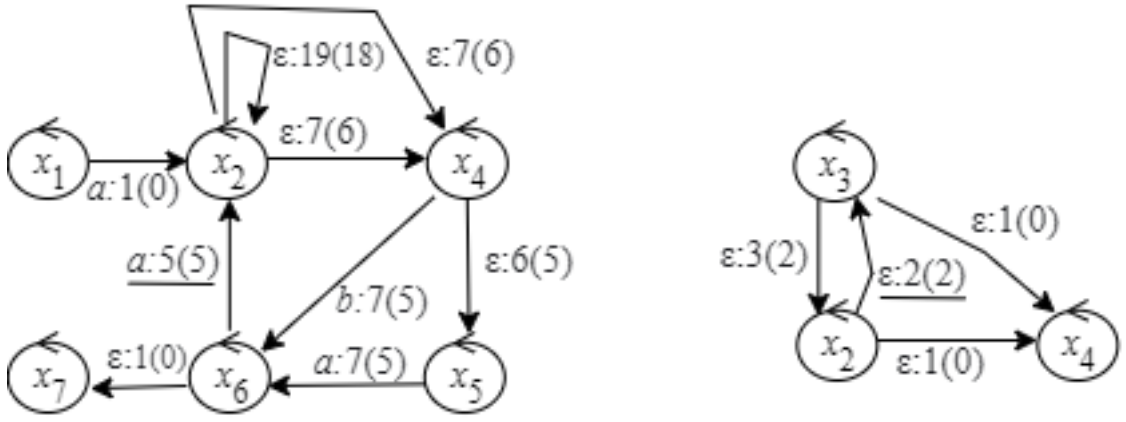
R, there is a corresponding activity profile $S' = S'(1), S'(2), ..., S'(n_2), n_2 \in IN$, that is derivable from R', where for every pair $(x, y) \in S(i)$ and $(y, z) \in S(i+1)$, either of the following holds:

1. $(x, y) \in S'(j)$ and $(y, z) \in S'(j+1)$,
2. there exists a path $p = x_1, x_2, ..., x_n \in R$ being represented by the arc (x_1, x_n) in R being represented by the arc (x_1, x_n) in R' where $x_{k-1} = x, x_k, x_{k+1} = z$, and
   (a) $(s, x_1) \in S'(j)$ and $(x_1, x_n) \in S'(j+1)$, for some vertex s in R',
   (b) $(x_1, x_n) \in S'(j)$, for some $1 \leq j \leq n_2$, or
   (c) $(x_1, x_n) \in S'(j)$ and x_n, $s \in S_{j-1}$, for some vertex s in R'.

Furthermore, we call (x_1, x_n) as the abstract arc in R' representing the path p in R.

Expanded Vertex Simplification

The process of vertex simplification results to an RDLT R lacking of L-values. Therefore, [8] proposes and extension of the process to produce *expanded vertex simplifiction* R'_1 and R'_2 with respect to R by using the RDLT and its simplified vertex to retrieve the other attributes, e.g. L. The algorithm for producing R'_1 and R'_2 is outlined in algorithm **??**, found in the appendices.



(a) Level 1 expanded vertex simplification

(b) Level 2 expanded vertex simplification

Figure 1.9: R'_1 and R'_2 expanded vertex simplification of the vertex simplified graphs in Figure 1.8

Soundness Property in RDLT

As explained in earlier sections, the existence of formalizations on the notions of soundness in Workflow nets served as the foundation for the formalizations of soundness on RDLT. Included in the list are classical and relaxed soundness, as formalized by [6], and weak and easy soundness by [11], all in the context of notions of soundness. The first two notions of soundness are defined below.

Definition 1.2.24. Classical Soundness of RDLTs [7] An RDLT is of classical sound if and only if the following requirement is satisfied by each 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 $\forall\ x \in I$ and $y \in V$, $(x, y) \in S(1)$, and $(u, f) \in S(k)$, $u \in V$:

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 $(x, y) \in S(k)$, $y = f$
   - If $k \leq 2$: 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 $(x, y) \in S[i]$, there exists another arc $(y, z) \in S(j)$, 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 $(x, y) \in E$, there is an activity profile $S' = \{S'(1), S'(2), ..., S'(k')\}$, where $(x, y) \in S'[i], 1 \leq k \leq k'$.

Based on this definition, the first condition requires an RDLT R's each specified vertex and their subsequent vertices leading to the final vertex of R. All required component of a JOIN is already resolved, or checked, by the activity extraction algorithm upon termination. In other words, an RDLT needs to have proper termination for each of its activity profiles. The second condition requires it to have no possible occurences of deadlocks for every component in the RDLT as it requires that all arcs are to be included in an activity profile during extraction, i.e., liveness of the RDLT. Figure 1.7 satisfies both conditions an is therefore classically sound.

Definition 1.2.25. Relaxed Soundness of RDLTs [6, 7, 13] An RDLT is of 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 $S = S(1), S(2), ..., S(k)$, where $1 \leq k \leq diam(R)$, where the following conditions hold:

- For every reachability configuration $S(t)$ prior to $S(k)$, if any, there should be at least one arc $(x, y) \in S(t)$ and another arc $(y, z) \in S(t')$, 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.
- The final reachability configuration $S(k)$ is only composed of arcs that point to the sink f, *i.e.* $\forall (x, y) \in S(k)$, $y = f$.
- The set of arcs traversed at a time t is strictly contained in the set of arcs traversed at time $t + 1$, *i.e.* $\bigcup_{i=1}^{t} S[i] \subset \bigcup_{i=1}^{t+1} S[i]$. Since an arc can occur in multiple $S[i]$, the operator $\bigcup$ should be considered a multiset union operator.
- Every arc $(x, y) \in E$ is traversed in some activity profile S', *i.e.* there exists an activity profile S', where $(x, y) \in S'(t)$ for some $S'(t) \in S'$.

In summary, relaxed soundness in the context of RDLT, follows the notion of relaxed soundness to that of a Workflow Net PN. It follows that the condition of proper

termination of a classically sound RDLT is relaxed or weakened, that is, for an RDLT to be of relaxed soundness it is only required that there be at least one activity profile where proper termination is obeserved, while still retaining the condition of liveness of a classically sound RDLT.

1.3 Problem Statement

The verification of soundness properties of RDLT can be done through the analysis of its L-safenesss, as done in [7]. Matrices can then be used, as done in the papers [5] and [4], to represent the RDLT information, e.g. the vertices and arcs, the attributes, the check/traversal components, and the entire state of the RDLT during execution up until execution. The matrix representation of an RDLT is then used for the verification of classical soundness and relaxed soundness in papers [4] and [5], respectively. In the context of this study, the notions of weak and easy soundness of RDLT was just recently formalized, along with the verification strategy, in the paper of [11]. This study is therefore centered on the matrix representation, and using related operations, to verify weak and easy soundness of RDLT, along with the creation of matrix-based data structures, relying on the methods and formalizations of the aforementioned studies. The lack of a well-defined matrix representation tailored to the verification of weak and easy soundness of RDLT, based on recent formalizations, presents a gap in the literature that this study aims to address.

1.4 Aim of the Work

General Objectives

Currently, there are already existing literature on weak and easy soundness in the context of RDLT. However, in the case of this research, the matrix representation for the verification of these notions of soundness is to be addressed. Therefore, the general objective of the study is the following:

1. To design a matrix based representation of RDLT, and related matrix operations, for the verification of weak and easy soundness; as well as to implement the verification algorithm of [11] for the verification of weak and easy soundness using matrix representation.

Specific Objectives

In alignment with the general objectives, the following specific objectives are identified to reach these aims:

For Weak Soundness Structural Verification

1. To establish matrix representation and operations for the verification of deadlock tolerance in both level 1 and level 2 expanded vertex simplification graphs [8], R_1 and R_2, respectively, of RDLT R [6].

2. To generate a list of deadlock points in R and identify escape contraction paths [11] [7].

3. To establish matrix representation and operations to verify weakened join-safe values of every split-join [11] pair in R.

4. To establish matrix representation and operations for the verification of weakened-join safeness [11] of R.

5. To verify the loop-safeness of NCA and safeness of CA [7].

6. To establish matrix representation and operations for the verification that R is deadlock-resolving [11].

7. To formulate test cases to prove the correctness of the proposed matrix represention and operations in verifying weak soundness of RDLTs [11].

8. To prove the correctness of the proposed matrix representation and operations in verifying weak soundness of RDLTs [11].

9. To measure the time and space complexity of the matrix-based verification of weak soundness of R.

For Easy Soundness Structural Verification

1. To formulate test cases to prove the correctness of the proposed matrix represention and operations in verifying easy soundness of RDLTs [11].

2. To prove the correctness of the proposed matrix representation and operations in verifying easy soundness of RDLTs [11].

3. To measure the time and space complexity of the matrix-based verification of easy soundness of R.

1.5 Scope and Limitations

The scope and limitations of the study are the following:

1. The study will focus on the design of matrix representation that encapsulates the structure of RDLT. It will incorporate matrix operations on the verification of the structural profiles of RDLT, as discussed in the paper of [11] to determine its weak and easy soundness. The study will not delve to alternative models other than matrix representation.

2. The study will center on RDLT as the main structural model to verify notions of soundness with [6]. Furthermore, its level 1 and level 2 expanded vertex simplified graphs will be used in the verification of weak soundness [8]. The study will not explore other types of models beyond these graphs.

3. The study will center on the verification of weak and easy notions of soundness [6] [11]. Its structural profiles will be used for the verification. The study does not include other notions of soundness.

4. The study will focus on verifying deadlock tolerance in the context of RDLT, as defined in the structural profile of weak soundness [11]. The study is limited to the definition of deadlock tolerance within this context.

5. The study aims to design and implement a matrix-based verification algorithms for weak and easy notions of soundness of an RDLT. The algorithms are only limited to weak and easy soundness and will not perform matrix operations beyond these verifications.

1.6 Significance of the Study

The study and its results will contribute to the enhanced understanding of RDLT and the verification of notions of soundness. Specifically, the study is centered on the matrix-representation of RDLT to verify weak and easy soundness, which could lead to improved methods for the analysis and modeling of complex systems. This deepens the theoretical framework of RDLT and strengthens its applicability in systems design. By establishing matrix-representation and operations for the verification of these notions of soundness, we address gaps and contirbute to the broader understanding of the field by:

1. Providing a structured framework for the verification of easy and weak soundness, which could lead to the enhancement of reliablity and accuracy of complex systems.

2. Introduce efficient deadlock detection and resolution mechanisms through matrix-based operations, leading to more robust systems.

3. Providing the groundwork for the advancement of automated verification tools, speeding up the validation process of RDLTs, in the scope of weak and easy notions of soundness.

1.7 Theoretical and Conceptual Framework

The Figure presents the conceptual framework of the research. It contains required concepts and definitions in order to proceed with the methodology and achieve the reseach objectives.

The main requirements for achieving the main objectives of the research are the following: definition of RDLT [6], definition of a weak and easy sound RDLT based on their structural and behavioral profiles [11], RDLT Activity extraction [6] [4], Vertex Simplification, and Matrix Operations [5]. RDLT is defined in [6] and the classical notion of soundness. By the scope of this research, the weak and easy notions of soundness are to be verified, the profiles and algorithm of which are already defined by [11]. Majority of the research's objectives is focused on the matrix-based verification of the deadlock tolerance (and its subsequent properties) of the input RDLT [11]. This methodology requires expanded vertex simplification [8] (both level-1 and level-2) and a proposed algorithm for activity extraction [6]. Finally, the final state vector output of the matrix operations on the input RDLT will be evaluated for its validity, then its weak and easy soundness.

The implementation of this research would therefore include the reusing of the definitions of RDLT [6], weak and easy soundness profiles [11], and the algorithm for the

verification thereof. The algorithm by [11] would be translated or modified to verify a matrix-based RDLT. To achieve these main objectives, specific objectives are to be achieved first, requiring the same set of concepts and algorithms. Finally, after formulating the test cases for verifying the correctness of the matrix-based verification of weak and easy soundness, the time and space complexity will also be performed.

Below is the concpetual framework Figure **??** which shows the concepts and components from related literature that are required to implement the objectives, displayed as yellow boxes, as well as the components for the implementations to satisfy the objectives of this study. This includes modified and novel methods, shown by orange and green boxes, respectively.

Chapter 2

Review of Related Literature

2.1 On Weak and Easy Notions of Soundness in RDLTs

As mentioned in earlier discussions, due to the dynamicity and freedom the RDLT model provides in expressing workflow in all of its dimensions, recent developments upon its formalizations have occured [6]. The concept of soundness in Workflow Nets, which implies that all of the every activity or work done by a system that can be determined in the model should exhibit proper termination and liveness, was extended to RDLT, as important of a property as it is. Although RDLT and the traditional workflow models (e.g., Workflow Nets, Business Process Management Notation, Petri Nets) differ on their syntax in presenting soundness, RDLT still adopts the conditions for soundness as discussed above.

As in traditional workflow models, there are multiple variations to the notion of soundness, aside from the classical notion: relaxed, weak, easy, and lazy soundness. Similarly, formalizations of the definition have occured in the literature of RDLT in recent years. In [7] classical and relaxed soundness were defined along with the formalisms and design strategies. Hence, proper termination and liveness were achieved in both structural and behavioral aspects of the activities an RDLT has. The absence of such definition in the context of weak and easy soundness became the focus in Ramirez' [11] paper.

On the conceptual basis of the paper, [11] initially explored the Workflow Nets, especially in the context of soundness. Weak and easy soundness, along with the other notions, in the context of workflow nets and petri nets were discussed. A weak sound RDLT requires for it to have the sink place o reachable from the source place i (option to complete), and the sink place o receives exactly one token and all other places must be empty (proper termination). In comparison to classical soundness, weak soundness is much relaxed as it does not require the involvement of every transition during execution [11]. Similarly, it is discussed that for an RDLT to be of easy sound, its sink place o should be reachable from source place i (option to complete).

This definition was then extended in[11] in the formalizations of weak and easy soundness in the context of RDLT, hence the following definitions.

Definition 2.1.1 (**Weak Soundness of RDLTs**). An RDLT R is of weak soundness if and only if the following requirement is satisfied by each activity profile $S = \{S(1), S(2), \ldots S(k)\}, 1 \leq k \leq diam(R), diam(R)$ is the diameter of R, of a set of source vertices $I \subset V$ and a final output vertex $f \in V$, where $\forall\ x \in I$ and $y \in V$, $(x, y) \in S(1)$, and $(u, f) \in S(k)$, $u \in V$:

1. **Proper termination.** For every activity profile $S = \{S(1), S(2), \ldots S(k)\}, 1 \leq k \leq diam(R)$, 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 $(x, y) \in S(k)$, $y = f$
   - If $k \leq 2$: 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 $(x, y) \in S(i)$, there exists another arc $(y, z) \in S(j)$, for all $1 \leq i < k$, and for some j in the range $i + 1 \leq j \leq k$.

Based on this definition, an RDLT is of weak soundness if it properly terminates, where every reached vertex in each activity profile of the RDLT leads to the sink. In the context of WF-Nets, this notion of soundness in RDLT is similar in a sense that it does not require the workflow to be live. A weak sound RDLT allows for arcs to not be traversed at all.
In figure2.1, a weak sound RDLT is shown. All the arcs leading to x_5 from x_4 can be traversed, except with the path starting from the arc (x_1, x_2), since it cannot be traversed.

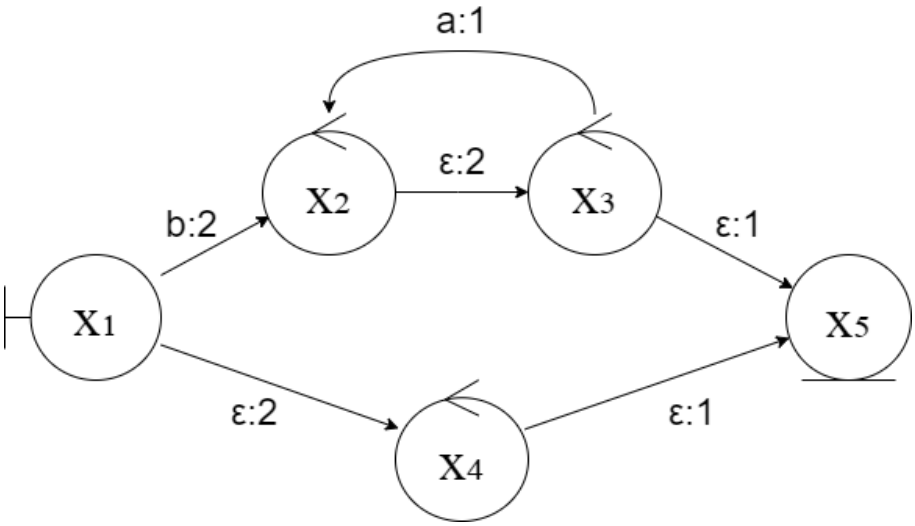


Figure 2.1: An RDLT of Weak Soundness (Image Source:[11])

The structural profile as defined by[11] of a weak sound RDLT is summarized by the following theorem:

Theorem 2.1.1. ***Structural Verification of the Weak Soundness of an RDLT*** *R [11] An RDLT R is weak-sound if and only if both the level-1 and level-2 expanded vertex simplifications of R, R_1 and R_2 respectively, are deadlock-tolerant.*

The algorithm**??** for the verification of weak soundness in RDLT is at the appendices.
On the verification of weak soundness in RDLT, it its stated at theorem2.1.1 that both expanded vertex simplifications of R, should be deadlock tolerant. To understand deadlock tolerance, the following definitons are stated. The following are sourced from [11] unless otherwse stated.

Definition 2.1.2. Weakened JOIN-safe RDLT R[11] An RDLT R is weakened JOIN-safe if every split-join pair in R has weakened JOIN-safe L-values.

Definition 2.1.3. Deadlock Point [11] A deadlock point is each vertex $x \in dV$ in an RDLT R where dV is a set of vertices in R and $dV \subset V$, such that $x \in dV$ is unreachable at some time step of the activity extraction algorithm.

Definition 2.1.4. Escape Contraction Path [11] An escape contraction path is a path $P = x_1, x_2 \dots x_n$ in an RDLT R such that the following holds:

1. x_1 is a parent of a deadlock point in R, i.e. $x_1 = p$ where $(p, q) \in E$ and $q \in dV$, and

2. P is a contraction path from x_1 to x_n in R where x_n is the sink vertex of R.

Definition 2.1.5. Deadlock-Resolving RDLT R [11] An RDLT R is deadlock-resolving if for every deadlock point $x \in V$, there exists at least one escape contraction path from a parent of x to the sink in R.

Lemma 2.1.2. *[11] An expanded vertex simplification R_i, where $i = 1, 2$, is deadlock-tolerant if and only if every activity thereof properly terminates.*

Definition 2.1.6. Deadlock-Tolerant RDLT R [11] An RDLT R is deadlock-tolerant if every NCA is loop-safe, every CA is safe, R is weakened JOIN-safe and deadlock-resolving.

Easy Soundness Notion of RDLTs

Definition 2.1.7. Self-Controlling Arc
[11] A self-controlling arc is a profile of an RDLT R where a MIX-JOIN or AND-JOIN at vertex x can never be resolved.

Figure 2.2, shown below, is an example of a self-controlling arc as defined in Definition 2.1.7. In this case, vertex b is the vertex that can never resolved due to one of the components of the AND-JOIN that violates the unconstrainedness criterion of the activity extraction algorithm.

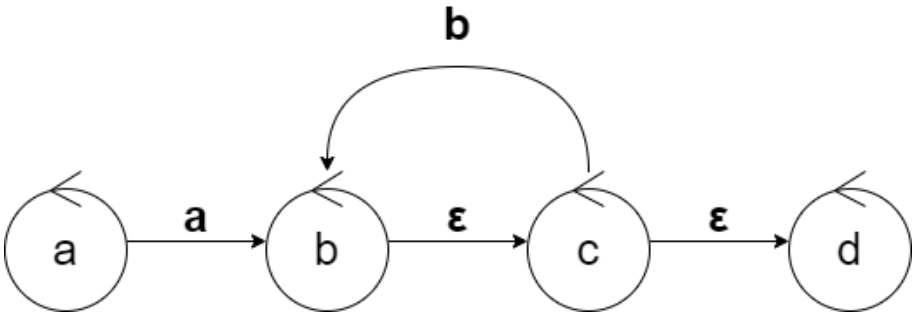
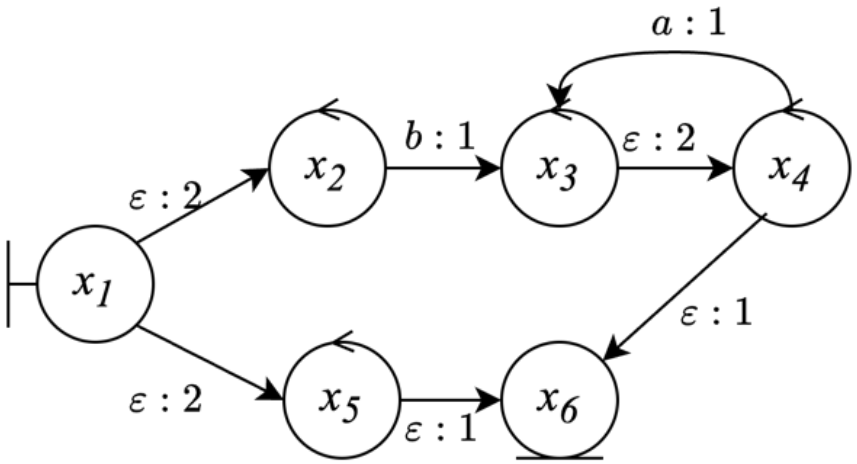


Figure 2.2: An RDLT a Self-Controllin Arc Image Source: [11]

Definition 2.1.8. Easy Soundness of RDLTs [11] An RDLT R is of easy soundness if and only if the following requirement is satisfied by an existing 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 $\forall\ x \in I$ and $y \in V$, $(x, y) \in S(1)$, and $(u, f) \in S(k)$, $u \in V$:

1. **Option to complete.** There exists an 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 $f \in V$ where there exists a path P composed of $P = x_1, x_2 \cdots x_n$ with $i = 1, 2 \cdots n-2$ where $(x_i, x_{i+1}) \in S(i), (x_{i+1}, x_{i+2}) \in S(j), i < j, x_i = s$ and $x_n = f$.



[11]

Figure 2.3: An RDLT of Easy Soundness

Based on this definition, the RDLT in Figure 2.3 is of easy sound.

Shown below is Theorem 2.1.3 that describes the structural requirements required to determine the easy soundness of an RDLT.

Theorem 2.1.3. ***Structural Verification of the Easy Soundness of an RDLT*** *R [11] An RDLT R is easy-sound if and only if the following elements exist:*

- *A contraction path P_1 in the level-1 vertex simplification R_1 of R from the initial vertex x_i to the final vertex x_f, wherein $P_1 = x_1, \ldots, x_f$*
- *A contraction path P_2 in the level-2 vertex simplification R_2 of R from the initial vertex z to the final vertex p of the RBS, wherein $P_2 = z, \ldots, p$*
- *An in-bridge (x, y) formed by a component in P_1 such that there exists a component (y, z) in P_2*
- *An out-bridge (q, p) formed by a component in P_2 such that there exists a component (p, r) in P_1*

2.2 On Verification Strategies of Soundness Properties of RDLT

Matrix Representation and Automation of Verification of Soundness of Robustness Diagram with Loop and Time Controls

In their paper titled *"Matrix Representation and Automation of Verification of Soundness of Robustness Diagram with Loop and Time Controls"* Karen and Roben [5] proposed a

matrix representation for the verification of soundness of RDLT. They used matrix representations for the state of the R during activity extraction. The verification of soundness, then, is done by analyzing the *final state vector* resulting from $\mathcal{A}$ they incorporated from [6]. Their methodology is shown to be effective in verifying *relaxed soundness* property of an RDLT. The following are the basic concepts and definitions necessary to understand the verification of soundness property of RLDT through matrix verification. The formal definition of these vectors and matrix operations are from [5] unless stated otherwise.

Definition 2.2.1. ***Counter Vector*** *[5]*
Let $R = (V, E, \Sigma, C, L, M)$ be an RDLT and $Q = (e_1, e_2, \ldots, e_q)$ be the finite sequence of arcs in R where $q = |E|$ and $e_i \in E$ is the i^{th} arc of the sequence. The counter vector at time step t is defined as

$$\alpha(t) = ({a_1}^{(t)}, {a_2}^{(t)}, \ldots, {a_q}^{(t)})$$

such that ${a_i}^{(t)} \in \mathbb{N}$ represents the total number of times arc e_i has been explored from time step 0 to time step t.

Definition 2.2.2. ***Constraint Vector*** *[5]*
Let $R = (V, E, \Sigma, C, L, M)$ be an RDLT and $Q = (e_1, e_2, \ldots, e_q)$ be the finite sequence of arcs in R where $q = |E|$ and $e_i \in E$ is the i^{th} arc of the sequence. The constraint vector at time step t is defined as

$$\beta(t) = ({b_1}^{(t)}, {b_2}^{(t)}, \ldots, {b_q}^{(t)})$$

such that

$${b_i}^{(t)} = \begin{cases} C(e_i)_y, & \text{if (arc } e_i \text{ is currently checked but cannot yet be traversed at time step } t) \text{ or (arc } e_i = (x, y) \text{ with } C(e_i) = \epsilon \text{ is not checked but there exists arc } e_j = (v, y) \text{ that is currently checked) where } C(e_i) \text{ is the attribute } C \text{ of arc } e_i; \\ -C(e_i)_y, & \text{if arc } e_i = (x, y) \text{ with } C(e_i) \in \Sigma \text{ is not checked but there exists arc } e_j = (v, y) \text{ that is currently checked where } C(e_i) \text{ is the attribute } C \text{ of arc } e_i; \\ 0_y, & \text{otherwise.} \end{cases}$$

Definition 2.2.3. A ***State Vector*** *[5]*
Let $\alpha(t)$ be the counter vector at time step t and $\beta(t)$ be the constraint vector at time step t. The state vector at time step t is defined as

$$\rho(t) = [\alpha(t), \beta(t)].$$

Aside from the vectors defined above, [5] also conceptualized the use of matrix to keep track of the steps done during activity extraction, that is the arcs explored at every time step t. These are the components for *activity extraction* using matrix representation:

Definition 2.2.4. ***Explored Arcs Vector***
Let $R = (V, E, \Sigma, C, L, M)$ be an RDLT and $Q = (e_1, e_2, \ldots, e_q)$ be the finite sequence of arcs in R where $q = |E|$ and $e_i \in E$ is the i^{th} arc of the sequence. The explored arcs vector at time step t is defined as

$$\gamma(t) = ({g_1}^{(t)}, {g_2}^{(t)}, \ldots, {g_q}^{(t)})$$

such that

$$g_i{}^{(t)} = \begin{cases} 1, & \textit{if arc } e_i = (x, y) \textit{ is chosen to be explored at time} \\ & \textit{step } t \textit{ where vertex } x \textit{ was already reached before} \\ & \textit{time step } t\textit{;} \\ 0, & \textit{otherwise.} \end{cases}$$

Definition 2.2.5. ***Incoming Arcs Vector***
Let $R = (V, E, \Sigma, C, L, M)$ be an RDLT, $Q = (e_1, e_2, \ldots, e_q)$ be the finite sequence of arcs in R where $q = |E|$ and $e_i \in E$ is the i^{th} arc of the sequence, and $\gamma(t) = (g_1{}^{(t)}, g_2{}^{(t)}, \ldots, g_q{}^{(t)})$ be the explored arcs vector at time step t. The incoming arcs vector at time step t is defined as

$$\phi(t) = (p_1{}^{(t)}, p_2{}^{(t)}, \ldots, p_q{}^{(t)})$$

such that

$$p_i{}^{(t)} = \begin{cases} 1, & \textit{if } (g_i{}^{(t)} = 1) \textit{ or } (g_i{}^{(t)} = 0 \textit{ and } C(e_i) = \epsilon_y \textit{ and} \\ & \exists g_j{}^{(t)} = 1) \textit{ where } e_i = (x, y), e_j = (v, y), \textit{ and } C(e_i) \\ & \textit{is the attribute } C \textit{ of arc } e_i\textit{;} \\ -1, & \textit{if } g_i{}^{(t)} = 0 \textit{ and } C(e_i) \in \Sigma \textit{ and } \exists g_j{}^{(t)} = 1 \textit{ where} \\ & e_i = (x, y), e_j = (v, y), \textit{ and } C(e_i) \textit{ is the attribute } C \\ & \textit{of arc } e_i\textit{;} \\ 0, & \textit{otherwise.} \end{cases}$$

Definition 2.2.6. ***C-Attribute Vector***
Let $R = (V, E, \Sigma, C, L, M)$ be an RDLT and $Q = (e_1, e_2, \ldots, e_q)$ be the finite sequence of arcs in R where $q = |E|$ and $e_i \in E$ is the i^{th} arc of the sequence. The C-attribute vector of R is defined as

$$\vec{C} = (c_1, c_2, \ldots, c_q)$$

such that $c_i = C(e_i)_y$ where $C(e_i)$ is the attribute C of $e_i = (x, y)$.

To describe R fully at each time step, the constraints of the arcs are also monitored, through an **unrefreshed constraint vector**, which at time step t shows which among the constraints of the arcs in R are satisfied. The formal definition is described below:

Definition 2.2.7. Unrefreshed Constraint Vector [5]
Let $\beta(t-1)$ be the constraint vector at time step $t-1$, $\phi(t)$ be the incoming arcs vector at time step t, and $\vec{C}$ be the C-attribute vector of R. The unrefreshed constraint vector at time step t is defined as

$$\omega(t) = \beta(t-1) \vee \big(\phi(t) \odot \vec{C}\big).$$

Similarly, The **refresh vector** is introduced to ensure that the constraints for reaching a vertex y are reset or refreshed once the vertex is reached.

Definition 2.2.8. Refresh Vector [5]
Let $\omega(t) = \big(w_1^{(t)}, w_2^{(t)}, \ldots, w_q^{(t)}\big)$ be the unrefreshed constraint vector at time step t. The refresh function of '$\omega(t)$ is defined as

$$\text{refresh}(\omega(t)) = Z$$

where $Z = \big(z_1, z_2, \ldots, z_q\big)$ such that

$$z_i = \begin{cases} 0_y, & \text{if } e_i = (x, y) \text{ is traversed at time step } t; \\ w_i, & \text{if } e_i \text{ cannot yet be traversed at time step } t. \end{cases}'$$

These vectors are supported by the matrix operations estabilished by [5], namely: **Literal Sign Function**, **Literal OR**, and **Literal Element-Wise Multiplication**.

Ultimately, these matrix representations and operations yield a final state vector, upon which properties of soundness can be analyzed, hence be verified.

Automated Verification of Classical Soundness in Robustness Diagrams with Loop and Time Controls via L-safeness

On their paper titled *"Automated Verification of Classical Soundness in Robustness Diagrams with Loop and Time Controls via L-safeness"*, Asoy [4] proposed a system for the automated verification of Classical Soundness of RDLT [7]. In this paper, they utilized the idea, and other defined vectors, of a Matrix-based verification of the propery of soundness, from the paper of Karen and Roben [5].

A summary of the methodology is as follows. An input RDLT is processed by the system to generate expanded vertex simplifications, either level-1 or level-2, depending on the existence of an RBS. Then, through L-safeness verification, the system performs generation of matrices and related matrix operations to verify the conditions of the L-safeness of the simplified RLDT's, of which's conclusion, if either classically sound or not, be generalized to the original input RLDT.

Asoy [4] was able to propose several algorithms that helped for the verificaion of the conditions of an L-safe RDLT. The following includes an algorithm for the generation of level-1 (or level-2) expanded vertex simplification, producing also *abstract arcs*, which is an arc that connects an R_2 vertex with an in-bridge or out-bridge, aggregating the l-attributes of tha arc across multiple arcs it is involved; an algorithm for determining the *loop-safeness* of an NCA; conversion of the expanded vertex simplfied graph to its corresponding matrix form, which is composed of the vectors C-attribute, L-attribute, eRU for all of the arcs; for the verification of *safeness of Critical Arcs*; and, finally, for the verification of L-safeness and Classical soundness of the simplified RDLT, hence the original input RDLT.

The vectors utilized in this system are: the first aforementioned vectors storing the C-attribute, L-attribute, and eRU of each arcs; *Cycle Vector*, which indicates whether an arc is a part of a cycle (1 if the arc is part of a cycle but not a critical arc, -1 if is a part of a cycle and is a critical arc, and 0 otherwise); the *Out Cycle Vector*, which indicates the existince of escape non-critical arc for a critical arc (C if a critical arc has an escape NCA, -C if it has an escape arc, but not NCA, and 0 otherwise); a vector for $loop - safeness$, which shows if an NCA can be reused as many times as it is required from all the cycles it is associated with, provied its value on the cycle vector is 1, or if its part of a cycle (1 if L-attribute is greater than eRU, -1 if L-attribute is less than eRU, 0 otherwise); vector for safeness of a Critical arc; and several others for the verification of the JOIN-safeness of RDLT.

Asoy [**?**] was able to utilize all of these vectors through a matrix and apply related operations for the verification of properties of an L-safe RDLT. On top of that, they were also able to generate the components of the RDLT upon which violations were detected during the execution of the aforementioned operations.

In summary, the utitlization of matrices in the verification of properties of an RDLT is an effective method, along with the consideration of the time and space complexity of the algorithms. In relation to this current research, the idea of using matrix-based verification of weak and easy soundness, just like in the verification of classical soundness in RDLT, is plausible and meritous idea, to which the study will be modifying to suit the structural characteristics of a weak and easy sound RDLT.

Chapter 3

Methodology

This section introduces a matrix-based implementation of the verfication of weak and easy soundness of RDLT. The general algorithm for these verification is based on the formalizations of [11] in terms of structural verifications of the said soundness properties.

Firstly, on weak soundness, [11] proposed WRSVA in which Deadlock-Tolerance is verified in order to verify weather or not an RDLT is weak sound. As explained in the related literatures, deadlock-tolerance refers to an RDLT being deadlock-resolving, having all NCAs loop-safe, CAs safe, and having all its split-join pairs as weakened JOIN-safe. All of these properties are verified through the use of matrices and their corresponding operations. The proofs and test-cases are also outlined for each verification of these mentioned properties.

Similarly, easy soundness verification as formalized by [11] is also implemented using matrix representation and operations. Its property of having an option to complete is verified by finding a contraction path from the source of R to the sink. Corresponding proofs and test-cases are also included to this proposed method.

3.0.1 Verification of Deadlock-Resolving RDLT

Theorem 3.4.3 in [11] explains that an RDLT R is of weak soundness if and only if both level-1 and level-2 vertex simplifications of R are deadlock-tolerance. The paper defines deadlock-tolerance as an RDLT where every NCA are loop-safe, every CA are safe, R is weakened JOIN-safe, and is deadlock-resolving.

For the verification of whether R is deadlock-resolving, the matrix representation of which includes the enumeration of relevant deadlocks in R, and the identification of the existence of escape contraction paths from the parent of each deadlocks. The existence of an escape contraction paths is what ensures proper termination albeit the existence of deadlocks within an RDLT, removing the liveness property. Therefore, such RDLT is of weak soundness.

Identification of Deadlocks

On the identification deadlocks, a method is proposed which involves the contraction [6] of the expanded vertex simplified RDLT R_i. This contraction algorithm as proposed by Malinao in here dissertation checks the c-verifiability of the RDLT [6]. Two adjacent vertices are contractable or can be merged when the arc currently being checked has contraint/s which is a subset of all the contraints coming towards the other vertex the algorithm is trying to contract. The final contraction shows the contraction path, which

is a path of vertcies, that are reachable through an activity, hence through activity extraction, as well. That said, no unreachable vertices will be included in the contracted vertices, hence deadlocks, as defined in [11], can be identified.

This paper implements a matrix-based contraction algorithm adopted from the modified contraction algorithm (MCA) introduced in [**?**]. The following sections define the necessary matrices and algorithms to achieve a research objective.

Definition 3.0.1. Adjacency Matrix RV_{adj}
Let $R_i = \big(V_i, E_i, \Sigma_i, C_i, L_i, M_i\big)$ be an expanded vertex simplification of an RDLT R. The $n \times n$ matrix of a directional graph, where $n = |V_i|$ and the rows and columns correspond to the vertex set $V_i = \{\, v_1, v_2, \ldots, v_n\}$ at time step t, is defined as

$$RV_{adj}^{t} \;=\; \begin{pmatrix} RV_{\text{adj}}(v_1, v_1) & RV_{\text{adj}}(v_1, v_2) & \cdots & RV_{\text{adj}}(v_1, v_n) \\ RV_{\text{adj}}(v_2, v_1) & RV_{\text{adj}}(v_2, v_2) & \cdots & RV_{\text{adj}}(v_2, v_n) \\ \vdots & \vdots & \ddots & \vdots \\ RV_{\text{adj}}(v_n, v_1) & RV_{\text{adj}}(v_n, v_2) & \cdots & RV_{\text{adj}}(v_n, v_n) \end{pmatrix}.$$

Each entry $RV_{adj}(x, y)$ represents the number of distinct arcs from x to y, given by

$$RV_{adj}(x, y) \;=\; \begin{cases} RV_{adj}(x, y) \in \mathbb{N}, & \text{if } (x, y) \in E_i \text{ in } R_i, \\ 0, & \text{otherwise.} \end{cases}$$

Definition 3.0.2. Constraints Matrix RV_C
We similarly define the matrix RV_c. The $n \times n$ matrix of a directional graph, where $n = |V_i|$ and the rows and columns correspond to the vertex set $V_i = \{\, v_1, v_2, \ldots, v_n\}$ at time step t, is defined as

$$RV_{C}^{t} \;=\; \begin{pmatrix} RV_c(v_1, v_1) & RV_c(v_1, v_2) & \cdots & RV_c(v_1, v_n) \\ RV_c(v_2, v_1) & RV_c(v_2, v_2) & \cdots & RV_c(v_2, v_n) \\ \vdots & \vdots & \ddots & \vdots \\ RV_c(v_n, v_1) & RV_c(v_n, v_2) & \cdots & RV_c(v_n, v_n) \end{pmatrix}.$$

Each entry $RV_C^t(x, y)$ represents the corresponding C-attribute of the arcs from x to y in the Adjacency Matrix, given by

$$RV_C^t(x, y) \;=\; \begin{cases} C(x, y) \in \Sigma \cup \{\epsilon\}, & \text{if } (x, y) \in E_i \text{ in } R_i, \\ \varnothing, & \text{otherwise.} \end{cases}$$

Algorithm 1 Matrix-based Contraction Path Generation

Given RDLT R; Pre-processing Steps:
Input: Expanded vertex simplification R_i of RDLT R
Output: Contraction Path P
Matrices: RV^t_{adj} and RV^t_{C}

Initialize C-Attribute Matrix RV^0_C of R_i
Let $s' \in V_i$ be the source and $f' \in V_i$ be the sink
Let $x = s'$
Initialize $P = \{x\}$
Let $t = 1$
while $\exists y \in V_i \mid RV^{t-1}_{\text{adj}}(x, y) \geq 1$ **do**
    $\mathcal{Y} \leftarrow \{y \in V_i \mid RV^{t-1}_{\text{adj}}(x, y) \geq 1\}$
    Select any $y \in \mathcal{Y}$
    Let LHS $= RV^{t-1}_C(x, y) \cup \{\epsilon\}$
    $\mathcal{U} \leftarrow \{u \in V_i \mid u \neq x \wedge (RV^{t-1}_{\text{adj}}(u, y) \geq 1)\}$
    Let RHS $= \bigcup_{u \in \mathcal{U}} RV^{t-1}_C(u, y)$
    if LHS $\supseteq$ RHS **then**
        Update $RV^{t-1}_C(u, y) = \epsilon, \forall (u, y) \in E_i, u \neq x$
        for all $u \in \mathcal{U}$ **do**
            $RV^{t-1}_C(u, y) = \epsilon$
        end for
        Let $z = x \wedge y$
        Let $z = xy =$ Matrix Addition of rows (columns) x and y in RV^{t-1}_{adj}
        for all $w \in V_i$ **do**
            RowMerge_Adj: $RV^t_{\text{adj}}(z, w) = RV^{t-1}_{\text{adj}}(x, w) + RV^{t-1}_{\text{adj}}(y, w)$
            ColMerge_Adj: $RV^t_{\text{adj}}(w, z) = RV^{t-1}_{\text{adj}}(w, x) + RV^{t-1}_{\text{adj}}(w, y)$
        end for
        Let $z = xy =$ Element-wise Set Union of rows (columns) x and y in RV^{t-1}_C
        for all $w \in V_i$ **do**
            RowMerge_C: $RV^t_C(z, w) = RV^{t-1}_C(x, w) \cup RV^{t-1}_C(y, w)$
            ColMerge_C: $RV^t_C(w, z) = RV^{t-1}_C(w, x) \cup RV^{t-1}_C(w, y)$
        end for
        $V_i = (V_i \setminus \{x, y\}) \cup \{z\}$
        Create RV^t_{adj} as an $m \times m$ matrix where $m = n - t$ and as the submatrix of RV^{t-1}_{adj} with rows and columns indexed by updated vertex set V_i of R_i
        Create RV^t_C as an $m \times m$ matrix where $m = n - t$ and as the submatrix of RV^{t-1}_C with rows and columns indexed by updated vertex set V_i of R_i
        Let $x = z$
        Let $P = P \cup \{y\}$
        Let $t = t + 1$
    end if
end while
return P

Algorithm 1 performs the first phase of MCA in [9] with some modification. Instead of doing the contraction until the sink, this algorithm 1 performs the contraction until there is no vertex that can be contracted, leaving the unreachable vertices as not part of

the merged vertices. This process involves checking the constraints coming from vertex x to y and verifying whether those set of constraints is a superset of all the constraints of arcs coming towards y. If so, then x and y can be contracted. With this procedure, $PointsofDelay$ (POD)[6] vertices will not be merged, at least if there is no looping arc towards those points which can resolve the constraints making them reachable at some time step t. PODs are vertices that cannot be visited immediately due to the unresolved constraints of its incoming arcs.

Verification of the Existence of Escape Conctraction Paths

After the identification of deadlocks in R, in order to verify whether or not it is deadlock resolving as per its definition, each deadlock should have an escape contraction path [**?**]. Additionally, the parents involved in this verification are only the reachable ones.

Then, the proposed algorithm will go through each parent of each deadlock to verify the existence of an escape contraction path.

Algorithm 2 Matrix-Based Finding Escape Contraction Path Algorithm

Given: RDLT R_i; Preprocessing
Input: Parent Vertex w, Sink f', Contraction Path P, Adjacency Matrix of R_i RV_{adj}
Output: Boolean, $True$, otherwise $False$

```
1: Initialize Visited[0...n − 1] ← false                ▷ for each vertex in V_i
2: Initialize Q ← empty queue
3: Q.queue(w)
4: Visited[w] ← true
5: while Q ≠ ∅ do
6:     current ← Q.dequeue()
7:     Let k be the index of current vertex in V_i
8:     if current = f' then
9:         return True
10:        for all vertex x in P do
11:            Let l be the index of x in V_i
12:            if RV_adj[k][l] = 1 ∧ Visted[l] = false then
13:                Q.enqueue(x)
14:                Visited[l] ← true
15:            end if
16:        end for
17:    end if
18: end while
19: return False
```

Algorithm 2 goes through the each parents of a deadlock point and finds an escape contraction path with it. A breadth-first search is used in this algorithm, and the escape contraction path is a path induced from the contaction path from the algorithm 1. It means that the escape contraction path is a path from the parent to the sink, and at the same time reachable, which is implied if there is a path that can be induced from the contraction path P. Algorithm 2 outputs a boolean, signifying whether or not there is an escape contraction path for the input deadlock.

Now that the deadlocks and escape contraction paths can be identified through the use of matrices and matrix operations, deadlock-resolving verification can now be implemented.

Algorithm 3 Matrix-based Deadlock-resolving Verification Algorithm

Input: Expanded Vertex Simplification R_i of RDLT R, Contraction Path Algorithm Generation 1, Find Escape Contraction Path Algorithm 2

Output: Boolean; $True$, otherwise $False$

Matrices: Adjacency Matrix RV_{adj}, Constraint matirx RV_C

$P \leftarrow ContractionPathGenerationAlgorithm(R_i)$ 1

Deadlock Identification: Vertices not in P, but are adjacent to P, and are PODs

$dV \leftarrow \emptyset$ ▷ Initialize deadlock set

$RV_{adj} \leftarrow RV^0_{adj}$

$RV_C \leftarrow RV^0_C$

$unreachable\_vertices \leftarrow V_i/P$

for all vertex $x \in unreachable\_vertices$ **do**

  Let l be the index of vertex x in V_i

  $constraints\_stack \leftarrow \emptyset$

  $constraints\_counter \leftarrow 0$

  for all row k in RV_C **do**

    $c \leftarrow RV_C[k][l]$

    if $c \neq \emptyset \wedge c \notin \{\epsilon\} \cup constraints\_stack$ **then**

      $constraints\_stack \leftarrow constraints\_stack \cup \{c\}$

      $constraint\_counter \leftarrow constraint\_counter + 1$

    end if

  end for

  if $constraints\_counter \geq 2$ **then**

    $dV \leftarrow x_l \cup dV$

  end if

end for

Enumeration of Parents of Deadlocks and Finding Escape Contraction Paths

for all vertex $x \in dV$ **do**

  $parents(x) \leftarrow \emptyset$

  Let l be the index of vertex x in V_i

  for all row k in RV_{adj} **do**

    if $RV_{adj}[k][l] = 1 \wedge x_k \in P$ **then** ▷ The parent vertex needs to be reachable, hence included in the contraction path

      $parent(x) \leftarrow x_k \cup parent(x)$

    end if

  end for

  for all parent vertex $w \in parent(x)$ **do**

    $escapePathExists \leftarrow findEscapePath(w, f', P, RV)$ 2

    if $escapePathExists = false$ **then**

      return $False$

    end if

  end for

end for

return $True$

This matrix-based verification of whether or not an RDLT R is deadlock-resolving or not involves the generation of contraction path, deadlock identification, parent enu-

meration for each deadlock, and verification of the existence of escape contraction path for each parent vertex. On deadlock identification, these are the vertices not part of the contraction path P and are not PODs. Using the matrices RV_{adj} and RV_C which is the adjacency matrix before any contraction deadlocks can be identified, which are the immediate PODs adjacent from the merged vertex through algorithm 1.

On the identification of the parents for each deadlocks, martix RV_{adj} is used as well as the contraction path P. Parent vertices used in finding contraction paths should be at least reachable, hence P is used.

To verify the existence of escape contraction path, there should be an existing path from the parent vertex w to the sink vertex f'. Using 2, it will return a boolean verifying whether or not a path can be induced from the contraction path P from the parent vertex w to the sink f'.

Formal algorithm for Weakened JOIN Safeness and Deadlock Tolerance to be added ASAP Matrix-based Easy Soundness Verification to be added ASAP Revisions and Edits from Ch 1 to 2 and Results and Discussion to be added ASAP

Bibliography

[1] Wil Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8:21–66, 02 1998.

[2] Willibrordus Martinus Pancratius van der Aalst. *Structural characterizations of sound workflow nets*, volume 9623 of *Computing Science Reports*. Technische Universiteit Eindhoven, Eindhoven, Netherlands, 1996.

[3] Willibrordus Martinus Pancratius van der Aalst, Kees Max van Hee, Arthur Harry Maria ter Hofstede, Natalia Sidorova, Henricus M.W. Verbeek, Marc Voorhoeve, and Moe Thandar Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23, 2011.

[4] Andrei Luz B. Asoy. Automated verification of classical soundness in robustness diagrams with loop and time controls via l-safeness, 2024.

[5] Roben D. Delos Reyes and Karen Margaret D. Agnes. Matrix representation and automation of verification of soundness of robustness diagram with loop and time controls, 2018.

[6] Jasmine A. Malinao. *On Building Multidimensional Workflow Models for Complex Systems Modelling*. PhD thesis, Fakultät für Informatik (Pattern Recognition and Image Processing Group) Institute of Computer Graphics and Algorithm, Technische Universität Wien, Vienna, Austria, 2017.

[7] Jasmine A. Malinao and Richelle Ann B. Juayong. Classical soundness in robustness diagram with loop and time controls. *Philippine Journal of Science 152(6B)*, pages 2327–2342, 2023.

[8] Jasmine A. Malinao and Richelle Ann B. Juayong. Reset profiles and classical soundness in robustness diagrams with loop and time controls. *Pre-proceedings of the 12th Workshop on Computation: Theory and Practice (WCTP 2023)*, pages 371–394, 2023.

[9] Jasmine A. Malinao and Richelle Ann B. Juayong. Model separability of robustness diagram with loop and time controls. *Science and Engineering Journal (SciEnggJ)*, 17(2):189–201, July–December 2024. Available online: August 29, 2024.

[10] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn, Germany, 1962.

[11] Ronnie M. Ramirez II. On weak, lazy, and easy soundness in robustness diagrams with loop and time controls, 2024.

[12] Therese Nuelle Roca and Jasmine A. Malinao. Well-handledness in robustness diagrams with loop and time controls. In *Lecture Notes in Computer Science (LNCS)*, 2024. Accepted at the 20th International Conference on Intelligent Tutoring Systems (ITS 2024). To appear in LNCS.

[13] Cris Niño N. Sulla and Jasmine A. Malinao. Mapping of robustness diagram with loop and time controls to petri net with considerations on soundness. In Katerina Kabassi, Phivos Mylonas, and Jaime Caro, editors, *Novel & Intelligent Digital Systems: Proceedings of the 3rd International Conference (NiDS 2023)*, pages 338–353, Cham, 2023. Springer Nature Switzerland.