Non-Recursive LSAWfP Models are Structured Workflows

Milliam Maxime Zekeng Ndadji, Franck Bruno Tonle Noumbo, Maurice Tchoupé Tchendji

Non-Recursive LSAWfP Models are Structured Workflows

M. M. ZEKENG NDADJI[*1,2], F. B. TONLE NOUMBO[1], M. TCHOUPÉ TCHENDJI[*1,2]

[1]Department of Mathematics and Computer Science, University of Dschang
[2]FUCHSIA Research Associated Team, https://project.inria.fr/fuchsia/

*E-mail : {ndadjimaxime, ttchoupe}@yahoo.fr

Abstract

Workflow languages play a key role in the discipline of Business Process Management (BPM): they allow business processes to be modelled in order to simplify their automatic management by means of BPM systems. Numerous workflow languages addressing various issues (expressiveness, formal analysis, etc.) have been proposed. In the last decade, some workflow languages based on context-free grammars (having then formal semantics) and offering new perspectives to process modelling, have emerged: LSAWfP (a Language for the Specification of Administrative Workflow Processes) is one of them. LSAWfP has many advantages over other existing languages, but it is its expressiveness (which has been very little addressed in previous works) that is studied in this paper. Indeed, the work in this paper aims to demonstrate that any non-recursive LSAWfP model is a structured workflow. Knowing that the majority of commercial BPM systems only implement structured workflows, the result of this study establishes that, although LSAWfP is still much more theoretical, it is a language with commercial potential.

Keywords

BPM; LSAWfP; Structured Workflows; Dyck Language; Serialization

I INTRODUCTION

In Business Process Management (BPM), process modelling is a key phase [1]. It is done using workflow languages and consists for a given process, in analysing it in order to define in a graphical way or by means of rules, its tasks and their execution order (this is called the process *control flow*), the actors in charge of executing these tasks and the data flow between the tasks: the result is called a workflow model or process model[1] [1]. Regarding the diversity of application domains and professional needs for process modelling, researchers have proposed a plethora of workflow languages; making it difficult to reach a consensus on which one to adopt [1]. Among the most significant workflow languages are BPMN (Business Process Model and Notation, considered as the de-facto workflow language) [2], WF-Net (Workflow Net) [3] and YAWL (Yet Another Workflow Language) [3]. In 2020, Zekeng et al. proposed the *Language for the Specification of Administrative Workflow Processes* (LSAWfP) [4] addressing several problems encountered by classical languages, notably: the absence of formal semantics, the use of processes as modelling units, the obtention of either non-executable specifications or

[1]Process and workflow are often used as synonyms: this is the case in this paper.

context-specific executable ones, etc. LSAWfP proposes to model the control flow of a given process using a grammatical model called *Grammatical Model of Workflow* (GMWf).

This work is interested in the expressiveness of LSAWfP from its control flow perspective. In the BPM domain, the study of the expressiveness of workflow languages is a common practice whose goal is to show their credibility; however, previous works on LSAWfP have paid too little attention to this aspect. The work in [4] shows that, for its control flow, LSAWfP supports the four basic routings, namely: sequential, parallel, alternative and iterative routings; but, the results of this study don't provide sufficient evidence to characterise the class(es) of workflows supported by LSAWfP. The contribution of this paper is to formally establish that LSAWfP models whose GMWf does not admit recursivity[2] (non-recursive LSAWfP models) are *structured workflows*. It is then established that the subclass of structured workflows that do not admit iteration can be modelled using LSAWfP.

The concept of structured workflow has been popularised in works published in the early 2000s [5] and has been the subject of several studies in the last two decades; it refers to a class of workflows for which several syntactic restrictions have been applied on the control flow. Throughout these works, several formalizations of structured workflows and identification of their properties were made. Furthermore, it was established that this class of workflows is supported by many commercial BPM systems (TIBCO BPM Enterprise[3], Signavio[4], Bizagi[5], SAP R/4HANA[6], etc.). The study carried out in this paper finds its relevance in showing that LSAWfP is expressive enough to be embedded in a commercial BPM system; and in this case, LSAWfP will provide a new and advantageous tool for designing workflow models, while preserving the already existing commercialized knowledge.

The rest of this paper is organized as follows: some basic concepts useful for the understanding of this paper are briefly presented in section II. The contribution is presented in section III. Sections IV and V are dedicated respectively to a discussion and conclusion.

II BACKGROUND

2.1 Some basic concepts

A workflow is generally composed of a collection of activities/tasks, a set of actors, and dependencies between activities. Activities correspond to individual steps in a business process, actors are responsible for the enactment of activities, and dependencies determine the execution sequence of activities and the data flow between them. From the control flow perspective, Kiepuszewski et al. [6] state that a workflow $\mathcal{W}$ consists of a set of process elements $\mathcal{P}$, and a transition relation $Trans \subseteq \mathcal{P} \times \mathcal{P}$ between elements. The set of process elements can be further divided into a set O_j of or-joins, a set O_s of or-splits, a set A_j of and-joins, a set A_s of and-splits, and a set $\mathcal{A}$ of activities.

Generally, the main purpose of a workflow language is to provide tools (graphical or not) to represent process elements and the relations between them. For example, the BPMN language

[2] Informally, a non-recursive grammar is a grammar in which there is no non-terminal symbol whose expansion by means of productions allows to obtain a string containing this same symbol.

[3] https://www.tibco.com/products/business-process-management

[4] https://www.signavio.com/

[5] https://www.bizagi.com/

[6] https://www.sap.com/products/s4hana-erp.html

[2] represents the activities (elements of $\mathcal{A}$) by rectangles, the elements of O_j, O_s, A_j, A_s by associated diamond shapes and their relations by arrows. One of the main criticisms of workflow languages is that, they permit an arbitrary composition of process elements when building workflow models. Indeed, this arbitrary character is illustrated by a lack of ordering during the aggregation of the different elements constituting the workflows [6]. In order to remedy this, Kiepuszewski et al. propose several concepts, in particular, that of *structured workflows*.

2.2 Structured workflows

2.2.1 Definition

A Structured Workflow (SW) is intuitively defined as a workflow in which, each or-split has a corresponding or-join and each and-split has a corresponding and-join [5]. This type of workflow guarantees significant properties: for example, a well-formed SW can't deadlock [5]. This class of workflows is one of the most requested by researchers in their formal analysis of workflows [7]. In a more formally way, Kiepuszewski et al [6] define a SW inductively as follows:

Definition 1: Structured Workflow (SW)

1. A workflow consisting of a single activity is a SW (**Single-activity pattern**).
2. Let X and Y be SWs. The concatenation of these workflows is a SW (**Sequence pattern**).
3. Let $X_1, \cdots, X_n$ be SWs, oj an or-join and os an or-split. The workflow with os as initial element, oj as final element, transitions between os and the initial elements of $\{X_i\}_{1\leq i\leq n}$ and, other transitions between the final elements of $\{X_i\}_{1\leq i\leq n}$ and oj, is then also SW (**Or pattern**).
4. Let $X_1, ..., X_n$ be SWs, aj an and-join and as an and-split. The workflow with as as initial element, aj as final element, transitions between as and the initial elements of $\{X_i\}_{1\leq i\leq n}$, and other transitions between the final elements of $\{X_i\}_{1\leq i\leq n}$ and aj, is then also SW (**And pattern**).
5. Let X and Y be SWs, oj an or-join and os an or-split. The workflow with oj as initial element, os as final element, transitions between oj and the initial element of X, between the final element of X and os, between os and the initial element of Y, and between the final element of Y and oj, is then also a SW (**Loop pattern**).

Figure 1 shows minimum SWs corresponding to the five patterns in definition 1.

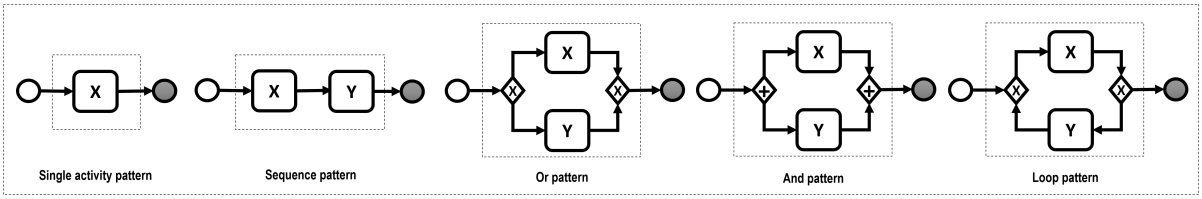

Figure 1: *Illustration of structured workflows patterns*

2.2.2 *Some formalizations of SWs*

Several studies have been focused on SWs during the last two decades; they proposed formal tools for the smooth handling of SWs. Jussi Vanhatalo et al. [8] have proposed to represent a structured workflow as a unique tree called Process Structure Tree (PST). This tree is obtained after the decomposition of the workflow into Single Entry Single Exit fragments. *A fragment of a given SW is a subset of its process elements that form a graph*. More precisely, the workflow is split into canonical fragments[7]; then, these canonical fragments are arranged to form a PST. The PST, computed in linear time, provides the necessary information about all the process elements, as well as the precedence relations between them. Its tree structure also allows formalizing operations such as well-structured and soundness verifications.

Another version of the PST called Redefined Process Structure Tree (RPST) has been proposed by Jussi Vanhatalo et al [7]. Unlike PST, RPST guarantees that a local change on the initial SW will only cause a local change on the resulting RPST. These tree-based formalizations of SWs gives us confidence in the grammatical approach of the work done in this paper. Indeed, in this paper, SWs are handled as Dyck words: i.e. as serializations of derivation trees for the grammar of the well-formed parenthesis language (Dyck language). It is this connection between SW and Dyck language that allows to establish the results presented in section III.

Besides PST and RPST, other studies [9, 10] have proposed Petri nets as a formal tool for handling SWs. Thanks to the mathematical character of Petri nets and to the numerous existing studies on their properties, they prove to be very useful for the study of the properties of SWs.

2.3 Non-recursive LSAWfP models

LSAWfP (a Language for the Specification of Administrative Workflow Processes) is a workflow language proposed by Zekeng et al [4]. In this one, the process control flow is modelled using a grammar $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ called **Grammatical Model of Workflow** (GMWf) in which :

- $\mathcal{S}$ is a finite set of **grammatical symbols** or **sorts** corresponding to various **activities** to be carried out in the studied process;
- $\mathcal{A} \subseteq \mathcal{S}$ is a finite set of particular symbols called **axioms**, representing activities that can start an execution scenario, and
- $\mathcal{P} \subseteq \mathcal{S} \times \mathcal{S}^*$ is a finite set of **productions** decorated by the annotations "⨟" (is sequential to) and "$\|$" (is parallel to): they are **precedence rules**. A production $P = \left(X_{P(0)}, X_{P(1)}, \cdots, X_{P(|P|)}\right)$ is either of the form $P : X_0 \rightarrow X_1 \fatsemi \cdots \fatsemi X_{|P|}$, or of the form $P : X_0 \rightarrow X_1 \parallel \cdots \parallel X_{|P|}$. The first form $P : X_0 \rightarrow X_1 \fatsemi \cdots \fatsemi X_{|P|}$ (resp. the second form $P : X_0 \rightarrow X_1 \parallel \cdots \parallel X_{|P|}$) means that activity X_0 must be executed before activities $\left\{X_1, \cdots, X_{|P|}\right\}$ that must be (resp. can be) executed in sequence (resp. in parallel) from the left to the right. A production with the symbol X as left-hand side is called a *X-production*. Given a production P, $|P|$ designates the length of its right-hand side.

The obtained specification after modelling a process with LSAWfP is called a LSAWfP model. A preliminary study of LSAWfP's expressiveness in [4], shows that it supports the basic routings pattern (sequential, parallel, alternative, iterative) in the definition of control flows. But, in this study, we are interested in a restricted form of LSAWfP models named *non-recursive LSAWfP models* that can be defined as follows:

[7] A canonical fragment is a fragment containing as few process elements as possible and whose combination corresponds to one of the patterns defined in [8], making it convertible into one or more elements of a PST.

Definition 2: Non-recursive LSAWfP model
A non-recursive LSAWfP model is a LSAWfP model whose GMWf (its grammar) is non-recursive.

We are interested in non-recursive LSAWfP models because they are the subclass of LSAWfP models in which activities are joined in a non-arbitrary way, using GMWf productions; this is actually their common point with structured workflows. Despite the fact that non-recursive LSAWfP models do not directly express iterative routing between process activities, they are useful in several practical cases; especially for administrative processes in which the recursivity (the number of repetitions) is generally bounded [11].

III SERIALIZING NON-RECURSIVE LSAWFP MODELS INTO WORDS OF A VERSION OF DYCK'S LANGUAGE DEDICATED TO LOOPLESS STRUCTURED WORKFLOWS SPECIFICATION

Let's remind that the goal of this paper is to establish that any non-recursive LSAWfP model is a SW. To achieve this, we will first consider a restriction of SWs class called *Loopless Structured Workflows (LSW)*; and, we will establish that the workflows in this new class can be assimilated to words of a version of Dyck's language that will be presented. Conversely, the words in this version of Dyck's language which we refer to as *Dyck's language for LSW* (denoted $Dyck^{LSW}$), are LSW. Finally, we will present a production rewriting algorithm to serialize any non-recursive GMWf into a word of $Dyck^{LSW}$ (i.e., into a SW through transitivity).

3.1 Loopless Structured Workflows (LSW)

Inspired by the notion of fragment used in the definition of PST and RPST (see section 2.2.2), we introduce the notion of *structured fragments*. A *structured fragment can be defined as a fragment corresponding to one of the five patterns in definition 1, allowing to recursively define SW*. Intuitively, a LSW is a SW in which no structured fragment matches the loop pattern of the SW definition (definition 1). Therefore, a LSW can be defined inductively as follows:

Definition 3: Loopless Structured Workflow (LSW)

1. A workflow consisting of a single activity is a LSW (**Single-activity pattern**).
2. Let X and Y be LSWs. The concatenation of these workflows is a LSW (**Sequence pattern**).
3. Let $X_1, \cdots, X_n$ be LSWs. The "Or pattern" as defined in definition 1, applied to $X_1, \cdots, X_n$, results to a LSW (**Or pattern**).
4. Let $X_1, \cdots, X_n$ be LSWs. The "And pattern" as defined in definition 1, applied to $X_1, \cdots, X_n$, results to a LSW (**And pattern**).

Given the only four patterns considered in the definition of LSWs, these can be expressed as words in a version of Dyck's language: the *Dyck's language for LSW* ($Dyck^{LSW}$). As a reminder, the Dyck language consists of strings of equal number of opening and closing brackets, and the number of closing brackets is never more than the opening brackets in any prefix of the string [12]. Indeed, if we follow definition 3, we can represent a single activity workflow by the following $Dyck^{LSW}$'s word : $\langle(_i)_i\rangle$. In this representation, $\langle$ and $\rangle$ represent the start and end events of the process and, $(_i)_i$ is a pair of colored parentheses representing the single activity being considered. The language $Dyck^{LSW}$ is denoted by the grammar of definition 4:

Definition 4: $Dyck^{LSW}$'s grammar
The grammar for the language $Dyck^{LSW}$ is defined by $\mathbb{G}_{Dyck^{LSW}} = (\mathcal{N}, \mathcal{T}, \mathcal{P}, Flow)$ where:

- $\mathcal{N} = \{Flow, Dyck\}$ is the set of non-terminals;
- $\mathcal{T} = \{\langle, \rangle, [_\vee,]_\vee, [_\wedge,]_\wedge\} \cup \{(_i,)_i\}_{1 \leq i \leq n}$ is the set of terminals; these are colored brackets that specify respectively, the start event, the end event, the or-split, the or-join, the and-split, the and-join and the different activities that make up processes;
- $\mathcal{P}$ is the set composed by the following productions:

$$\begin{array}{lcl} Flow & \longrightarrow & \langle Dyck \rangle \\ Dyck & \longrightarrow & (_i)_i \; Dyck \mid [_\vee Dyck]_\vee \mid [_\wedge Dyck]_\wedge \mid \epsilon \end{array}$$

- $Flow$ is the axiom.

The above grammar can be refined to avoid denoting uninteresting words such as $\langle \; \rangle$, $\langle [_\vee]_\vee \rangle$, etc. This is also done in order to group blocks, in classical parentheses to avoid ambiguity when necessary. The new productions of $\mathbb{G}_{Dyck^{LSW}}$ are the following:

$$\begin{array}{lcl} p_1 : Flow & \longrightarrow & \langle Frag \; NextFrag \rangle \\ p_2 : Frag & \longrightarrow & Seq \mid Or \mid And \\ p_3 : Seq & \longrightarrow & (_i)_i \; NextSeq \\ p_4 : NextSeq & \longrightarrow & Seq \mid \epsilon \\ p_5 : Or & \longrightarrow & [_\vee Frag \; NextFrag]_\vee \mid [_\vee (Frag) \; NextFrag]_\vee \\ p_6 : And & \longrightarrow & [_\wedge Frag \; NextFrag]_\wedge \mid [_\wedge (Frag) \; NextFrag]_\wedge \\ p_7 : NextFrag & \longrightarrow & Frag \mid (Frag) \mid \epsilon \end{array}$$

One can observe from the productions p_5 and p_6 of $\mathbb{G}_{Dyck^{LSW}}$, that in any word of $Dyck^{LSW}$, to each or-split (resp. and-split) corresponds an or-joint (resp. and-joint). Moreover, the productions p_1, p_2, p_3 and p_4 show that the simplest words accepted by $\mathbb{G}_{Dyck^{LSW}}$ are of the form $\langle (_i)_i \rangle$: they are LSWs consisting of a single activity. In the end, $\mathbb{G}_{Dyck^{LSW}}$ allows to recognize words made of (loopless) structured fragments combined only by means of the operators $CONCAT$, OR and AND defined as follows:

Definition 5:
Let $\langle w_1 \rangle, \langle w_2 \rangle, \langle w_3 \rangle, \cdots, \langle w_n \rangle$ be $Dyck^{LSW}$'s words (LSWs) :

1. The concatenation of these words is done using the operator $CONCAT$ that acts as follows: $CONCAT(\langle w_1 \rangle, \langle w_2 \rangle, \langle w_3 \rangle, \cdots, \langle w_N \rangle) = \langle w_1 w_2 w_3 \cdots w_n \rangle$ The operator $CONCAT$ helps to reproduce the sequence pattern of definition 3.
2. The or pattern (definition 3) is reproduced using the operator OR defined by:
   $OR(\langle w_1 \rangle, \langle w_2 \rangle, \langle w_3 \rangle, \cdots, \langle w_N \rangle) = \langle [_\vee (w_1)(w_2)(w_3) \cdots (w_n)]_\vee \rangle$ or
   $OR(\langle w_1 \rangle, \langle w_2 \rangle, \langle w_3 \rangle, \cdots, \langle w_N \rangle) = \langle [_\vee w_1 w_2 w_3 \cdots w_n]_\vee \rangle$ when there is no ambiguity.
3. The and pattern (definition 3) is reproduced using the operator AND defined by:
   $AND(\langle w_1 \rangle, \langle w_2 \rangle, \langle w_3 \rangle, \cdots, \langle w_N \rangle) = \langle [_\wedge (w_1)(w_2)(w_3) \cdots (w_n)]_\wedge \rangle$ or
   $AND(\langle w_1 \rangle, \langle w_2 \rangle, \langle w_3 \rangle, \cdots, \langle w_N \rangle) = \langle [_\wedge w_1 w_2 w_3 \cdots w_n]_\wedge \rangle$ when there is no ambiguity.
4. In addition, $CONCAT(\langle w \rangle)$, $OR(\langle w \rangle)$ and $AND(\langle w \rangle)$ are identities:
   i.e. $CONCAT(\langle w \rangle) = OR(\langle w \rangle) = AND(\langle w \rangle) = \langle w \rangle$

Apart from the words respecting the patterns presented above, the grammar $\mathbb{G}_{Dyck^{LSW}}$ cannot recognize other words. It can therefore only recognize LSWs written with colored parentheses (the serialized form of these LSWs): i.e. any word conforming to this grammar is a LSW.

3.2 Non-recursive LSAWfP models serialization to Dyck words

Having established that any LSW can be serialized as a $Dyck^{LSW}$'s word and that any word in the $Dyck^{LSW}$ language is a LSW, the second part of this paper's contribution consists in showing how to serialize any non-recursive LSAWfP model (its GMWf) into a $Dyck^{LSW}$'s word.

3.2.1 The serialization principle

To serialize a given GMWf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ into a $Dyck^{LSW}$'s word, one should proceed by successive rewritings of the right-hand sides of productions such as to obtain pieces of $Dyck^{LSW}$'s words (more precisely, $Dyck^{LSW}$'s words stripped of their start and end events (their symbols)) in these right-hand parts. This rewriting is done according to the principle described below :

- Productions of the form $X \longrightarrow \epsilon$ (epsilon productions) are considered rewritten;
- A production p of the form $p : X_0 \rightarrow X_1 \fatsemi \cdots \fatsemi X_{|p|}$ is rewritten as follows :

$$X_0 \longrightarrow \_CONCAT\left(\left\{(_i)_i \; \_OR\left(rhs\left(RP_{X_{i1}}\right), \cdots, rhs\left(RP_{X_{im_i}}\right)\right)\right\}_{1\leq i\leq |p|}\right)$$

  In this rewriting, the $\left\{RP_{X_{ij}}\right\}_{1\leq i\leq |p|;1\leq j\leq m_i}$ are the rewritten versions of the X_i-productions of the GMWf $\mathbb{G}$, and rhs designates their right-hand sides (thus pieces of $Dyck^{LSW}$'s words). The rewriting therefore produces a piece of $Dyck^{LSW}$'s word in the right side.
- Analogously, a production p of the form $p : X_0 \rightarrow X_1 \parallel \cdots \parallel X_{|p|}$ is rewritten :

$$X_0 \longrightarrow \_AND\left(\left\{(_i)_i \; \_OR\left(rhs\left(RP_{X_{i1}}\right), \cdots, rhs\left(RP_{X_{im_i}}\right)\right)\right\}_{1\leq i\leq |p|}\right)$$

After rewriting all productions, the serialized version ($Dyck^{LSW}$'s word) of the GMWf $\mathbb{G}$ is obtained as follows:

$$Ser(\mathbb{G}) \; = \; \left\langle \_OR\left(\left\{(_{A_i})_{A_i} \_OR\left(rhs\left(RP_{A_{i1}}\right), \cdots, rhs\left(RP_{A_{im_i}}\right)\right)\right\}_{1\leq i\leq |\mathcal{A}|}\right)\right\rangle$$

where the $\left\{RP_{A_{ij}}\right\}_{1\leq i\leq |\mathcal{A}|;1\leq j\leq m_i}$ are the rewritten versions of A_i-productions and the $\{A_i \in \mathcal{A}\}$ are the axioms of G. Note that in these rewritings, we use the $\_CONCAT$, $\_OR$ and $\_AND$ operators, respectively analogous to $CONCAT$, OR and AND, but which act (similarly to their counterparts) on $Dyck^{LSW}$'s words stripped of their start and end events (pieces of $Dyck^{LSW}$'s words).

3.2.2 An illustrative example

To better illustrate the presented concepts, let us take as an application case, the serialization of the LSAWfP model coming from the running example of [13]. The considered GMWf $\mathbb{G} = (\mathcal{S}, \mathcal{P}, \mathcal{A})$ is the one describing a peer review process. In it, $\mathcal{S}$ (the set of activities) is given by $\mathcal{S} = \{A, B, C, D, S1, E1, E2, F, G1, G2, H1, H2, I1, I2\}$, $\mathcal{A}$ (the set of axioms) is given by $\mathcal{A} = \{A\}$ and, the productions are the ones listed in the leftmost column of table 1. By applying the rewrite principle described in section 3.2.1, one should obtain the rewritten productions listed in the rightmost column of table 1. The serialized version of the GMWf $\mathbb{G}$ is finally obtained by applying the formula presented in section 3.2.1, and its value is as follows:
$Ser(\mathbb{G}) = \langle (_A)_A[_\vee(\ (_B)_B\ (_D)_D\ )\ (\ (_C)_C(_E)_E\ [_\wedge(\ (_{G1})_{G1}(_{H1})_{H1}(_{I1})_{I1}\ )\ (\ (_{G2})_{G2}(_{H2})_{H2}(_{I2})_{I2}\ )]_\wedge\ (_F)_F(_D)_D\ )]_\vee\rangle$
Figure 2 gives a graphical representation of the structured workflow described by the obtained $Dyck^{LSW}$'s word.

Productions	Rewritten Productions
$P_1 :\ A \to B \fatsemi D$	$RP_1 :\ A \to (_B)_B\ (_D)_D$
$P_2 :\ A \to C \fatsemi D$	$RP_2 :\ A \to (_C)_C(_E)_E\ [_\wedge(\ (_{G1})_{G1}(_{H1})_{H1}(_{I1})_{I1}\ )\ (\ (_{G2})_{G2}(_{H2})_{H2}(_{I2})_{I2}\ )]_\wedge\ (_F)_F(_D)_D$
$P_3 :\ C \to E \fatsemi F$	$RP_3 :\ C \to (_E)_E\ [_\wedge(\ (_{G1})_{G1}(_{H1})_{H1}(_{I1})_{I1}\ )\ (\ (_{G2})_{G2}(_{H2})_{H2}(_{I2})_{I2}\ )]_\wedge\ (_F)_F$
$P_4 :\ E \to G1 \parallel G2$	$RP_4 :\ E \to [_\wedge(\ (_{G1})_{G1}(_{H1})_{H1}(_{I1})_{I1}\ )\ (\ (_{G2})_{G2}(_{H2})_{H2}(_{I2})_{I2}\ )]_\wedge$
$P_5 :\ G1 \to H1 \fatsemi I1$	$RP_5 :\ G1 \to (_{H1})_{H1}\ (_{I1})_{I1}$
$P_6 :\ G2 \to H2 \fatsemi I2$	$RP_6 :\ G2 \to (_{H2})_{H2}\ (_{I2})_{I2}$
$P_7 :\ B \to \varepsilon$	$RP_7 :\ B \to \varepsilon$
$P_8 :\ D \to \varepsilon$	$RP_8 :\ D \to \varepsilon$
$P_9 :\ F \to \varepsilon$	$RP_9 :\ F \to \varepsilon$
$P_{10} :\ H1 \to \varepsilon$	$RP_{10} :\ H1 \to \varepsilon$
$P_{11} :\ I1 \to \varepsilon$	$RP_{11} :\ I1 \to \varepsilon$
$P_{12} :\ H2 \to \varepsilon$	$RP_{12} :\ H2 \to \varepsilon$
$P_{13} :\ I2 \to \varepsilon$	$RP_{13} :\ I2 \to \varepsilon$

Table 1: GMWf productions and their rewritten versions

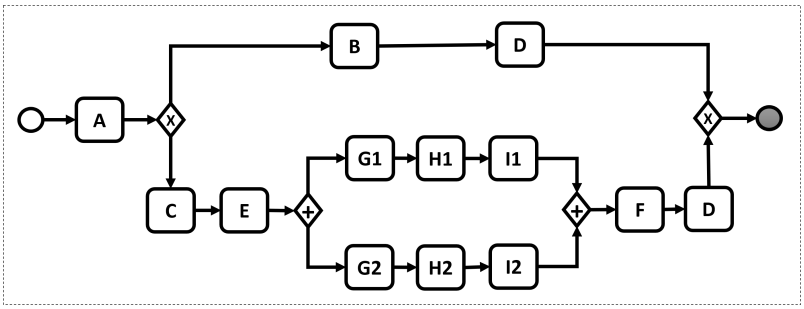

Figure 2: *Workflow diagram corresponding to our GMWf*

IV DISCUSSION AND FURTHER WORK

The work done and presented in this paper has established that a class of workflows modelled with LSAWfP is equivalent to the class of LSWs. The results obtained reinforce the idea that LSAWfP language is defined on a solid formal basis that can facilitate the study of the properties of LSAWfP models. Most of the existing studies like [14], aiming at showing the equivalence between two classes of workflows specified in two different languages, proceed by converting models from one language to the other. This is what has been done in this paper with the particularity that, it was first necessary to find a formal mean (the language $Dyck^{LSW}$) to specify structured workflows. The methodology used in this paper could be strengthened by proofs of several properties of the manipulated mathematical tools. Moreover, the paper was only interested in a sub-language of the LSAWfP language (the non-recursive LSAWfP models); an extension of the work done here to the whole LSAWfP language would certainly be more beneficial.

The result presented here opens the way to several potential works, notably: the conversion of LSAWfP specifications into classical formats such as those of the BPMN language, the verification of LSAWfP models and the use of a version of the Dyck language as a workflow language.

V CONCLUSION

In this paper, we have conducted some formalization studies in order to show the equivalence between a subset of structured workflows (LSW) and a subset of workflows that can be modelled with the LSAWfP language (non-recursive LSAWfP models). We used a variant of Dyck's language as an intermediate mathematical tool between the two manipulated subsets. The results of this paper help to promote LSAWfP by revealing a little more its commercial potential

and, offers some interesting perspectives in the analysis of its expressiveness. Immediate research avenues that could be of interest to potential researchers are: analysis and verification of LSAWfP specifications, conversion of an LSAWfP specification into a BPMN specification, integration of LSAWfP as a process modelling method in various commercial BPM systems.

REFERENCES

[1] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers. *Fundamentals of Business Process Management, Second Edition*. Springer, 2018. ISBN: 978-3-662-56508-7.

[2] B. P. Model. "Notation (BPMN) version 2.0". In: *OMG Specification, Object Management Group* (2011), pages 22–31.

[3] W. M. P. Van Der Aalst and A. H. M. Ter Hofstede. "YAWL: yet another workflow language". In: *Information systems* 30.4 (2005), pages 245–275.

[4] M. M. Zekeng Ndadji, M. Tchoupé Tchendji, C. Tayou Djamegni, and D. Parigot. "A Language and Methodology based on Scenarios, Grammars and Views, for Administrative Business Processes Modelling". In: *ParadigmPlus* 1.3 (2020), pages 1–22.

[5] B. Kiepusewski. "Expressiveness and suitability of languages for control flow modelling in workflows". PhD thesis. Queensland University of Technology, Brisbane, 2003.

[6] B. Kiepuszewski, A. H. M. ter Hofstede, and W. M. P. van der Aalst. "Fundamentals of control flow in workflows". In: *Acta Informatica* 39.3 (2003), pages 143–209.

[7] J. Vanhatalo, H. Völzer, and J. Koehler. "The refined process structure tree". In: *Data & Knowledge Engineering* 68.9 (2009), pages 793–818.

[8] J. Vanhatalo, H. Völzer, and F. Leymann. "Faster and more focused control-flow analysis for business process models through sese decomposition". In: *International Conference on Service-Oriented Computing*. Springer. 2007, pages 43–55.

[9] N. R. Adam, V. Atluri, and W.-K. Huang. "Modeling and analysis of workflows using Petri nets". In: *Journal of Intelligent Information Systems* 10.2 (1998), pages 131–158.

[10] D. Liu, J. Wang, S. C. Chan, J. Sun, and L. Zhang. "Modeling workflow processes with colored Petri nets". In: *computers in industry* 49.3 (2002), pages 267–281.

[11] M. M. Zekeng Ndadji. "A Grammatical Approach for Distributed Business Process Management using Structured and Cooperatively Edited Mobile Artifacts". PhD thesis. University of Dschang, Cameroon, 2021.

[12] X. Yu, N. T. Vu, and J. Kuhn. "Learning the Dyck language with attention-based Seq2Seq models". In: *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 2019, pages 138–146.

[13] M. M. Zekeng Ndadji, M. Tchoupé Tchendji, C. Tayou Djamegni, and D. Parigot. "A projection-stable grammatical model for the distributed execution of administrative processes with emphasis on actors' views". In: *Journal of King Saud University - Computer and Information Sciences* (2021). ISSN: 1319-1578.

[14] S. Pornudomthap and W. Vatanawood. "Transforming YAWL workflow to BPEL skeleton". In: *2011 IEEE 2nd International Conference on Software Engineering and Service Science*. 2011, pages 434–437.