Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Assessing SMT and CLP Approaches for Workflow Nets Verification

Hadrien Bride[1,2], **Olga Kouchnarenko**[1], **Fabien Peureux**[1], **Guillaume Voiron**[1]

[1] Institut FEMTO-ST – UMR CNRS 6174, Univ. Bourgogne Franche-Comté
16, route de Gray, 25030 Besançon, France
e-mail: {hbride,okouchna,fpeureux,gvoiron}@femto-st.fr

[2] Ecole Centrale de Nantes, LS2N, UMR CNRS 6004
1 Rue de la No, 44300 Nantes, France
e-mail: hbride@ls2n.fr

The date of receipt and acceptance will be inserted by the editor

Abstract. In the actual business world, companies rely more and more on workflows to model the core of their business processes. In this context, the focus of workflow analysts is made on the verification of workflows specifications, in particular of modal specifications that allow the description of necessary or admissible behaviours. The design and the analysis of business processes commonly relies on workflow nets, a suited class of Petri nets. The goal of this paper is to evaluate and compare in a deep way two resolution methods—Satisfiability Modulo Theory (SMT) and Constraint Logic Programming (CLP)—applied to the verification of modal specifications over workflow nets. Firstly, it provides a concise description of the verification methods based on constraint solving. Secondly, it introduces the toolchain developed to automate the full verification process. Thirdly, it describes the experimental protocol designed to evaluate and compare the scalability and efficiency of both resolution approaches and reports on the obtained results. Finally, these obtained results are discussed in detail, lessons learned from these experiments are given, and, on the basis of experiments feedback, directions for improvement and future work are suggested.

Key words: Workflow nets, Modal specifications, Verification method, Experimental comparison, Satisfiability Modulo Theory, Constraint Solving Problem

1 Introduction

In recent years, the growing need by companies to improve their organizational efficiency and productivity has led to the design and the analysis of business processes. Major Key Performance Indicators (compliance with respect to regulations and directives, end-user acceptance and confidence, etc.) are often directly determined by the quality of the business process in use, and therefore much of the companies successes depends on them. In order to control such business processes and to manage their tasks and steps, a great diversity of application domains uses workflow management systems on a daily basis. These include office automation, healthcare,telecommunication, manufacturing and production, finance and banking, just to name a few.

Workflows actually constitute a convenient way for analysts to describe the business processes in a formal and graphical manner. Nowadays they are thus extensively used by the economic and scientific communities to model and analyse processes. Intuitively, a workflow system describes the set of possible runs of a particular system/process. Due to the critical relation between the quality of the workflows in use and the success of the companies in terms of efficiency, productivity and security, workflow specification verification has thus become one of the major activities of workflow specialists. Furthermore, workflow analysts are required to express and to verify specific properties over the workflows they designed to ensure the validity of the related process and make sure that no undesirable behaviour is present while performing the specified tasks.

Among existing workflow specifications, this paper focuses on modal specifications that allow the description of necessary and admissible behaviours over workflow nets, a suited class of Petri nets. Working on case studies of real-life instances of industrial workflows obtained through collaborations with industrial actors, described in part in [3,4], has emphasized the importance of expressing and verifying—at the specification or design stage of the development of large-size workflow nets—some required behavioural properties, derived from textual requirements and business analyst expertise. Further, these collaborations have demonstrated the limited scalability of traditional model checking tools based on state space exploration when verifying properties such as the behavioural properties expressed by modal specifications, thereby, assessing the need for efficient and scalable alternative verification methods. To this end, a recent work in [3] and [4] has introduced and developed a novel formal framework based on constraint systems to model executions of workflow nets and their structural properties, as well as to verify requirements specified by modal specifications.

One of the advantages of such a method is the use of constraint-based models allowing the proposed verification method to benefit from existing mature and efficient constraint solvers as they assume most of the computational workload. More precisely, in this framework, the validity of a modal specification can be inferred from the satisfiability of a corresponding Constraint Satisfaction Problem (CSP), which can be solved using Constraint Logic Programming (CLP) or using Satisfiability Modulo Theory (SMT) solvers.

On the one hand, using Logic Programming for solving a CSP has been investigated for many years, especially using CLP over Finite Domains, written CLP(FD). This approach basically consists in embedding consistency techniques [12] into Logic Programming by extending the concept of logical variables to the one of the domain-variables taking their values in a finite discrete set of integers. On the other hand, SMT solvers are also relevant to solve the constraint systems (a conjunction of boolean formulas expressing the constraints) since they can determine whether a first-order logic formula can be satisfied with regards to a particular theory (e.g., Linear Arithmetic, Arrays theories). SMT solvers aim to generate counter-examples [11] by combining a SAT solver, assigning a truth value to every atom composing the formula so that the truth value of the latter is true, with a theory solver determining whether the resulting interpretation can be met with regard to the theory used. The formula is satisfiable if and only if at least one interpretation from the SAT solver can be met by the theory solver.

Besides the theoretical assessment of the approach, a proof-of-concept toolchain implementing this approach has enabled to successfully evaluate its effectiveness and reliability. However, as advocated in [4,7], these first encouraging experimental results need to be confirmed by extensive and further experimentation, in particular to definitively assess the scalability and the efficiency of the approach. This paper precisely investigates these issues. Therefore, the main contributions of this paper are the following:

(1) an empirical assessment of the scalability of the verification approach proposed in [3] and [4],
(2) an accurate study to evaluate the efficiency of this approach by experimenting two resolution methods –SMT and CLP over Finite Domains– to solve the constraint system that represents the modal specifications to be verified.
(3) a comparison of the both above-mentioned resolution methods to evaluate their relevance and efficiency within this verification approach.

Notice that the contributions of the present article are based on formal means first introduced in [3] and further developed in [4]. As extension to [6,7], this paper provides a full description of the used tools, together with a finer and larger experimental basis. This new basis, composed of 6400 workflow nets containing up to 1000 nodes, allows us to better address issues (1) and (2). Differently from previous work [6,7], where CLP computation was performed without labeling heuristics, the new CLP results are computed using the strategy *First Fail Cut*. It provides the more conclusive and convincing results in comparison with the other CLP heuristics implemented by SICStus Prolog, which have also been experimented but are not shown and discussed in the present paper.

As a consequence, this paper reports on the obtained experimental results, which are important and completely new. Consequently, with relation to issue (3), the already learned lessons are revisited and new lessons are pointed out.

Layout of the paper. Section 2 briefly recaps common concepts and standard notations concerning workflow nets, while Sect. 3 introduces the key aspects of the formal method given in [3] for verifying modal specifications over workflow nets. Afterwards, Sect. 4 describes the toolchain implementing this method: from a workflow net and its modal specification, it automatically produces a corresponding constraint system whose satisfiability can then be checked using either CLP or SMT computation. Section 5 first defines the experimental protocol designed, on the one hand, to evaluate the efficiency of each resolution approach, and, on the other hand, to compare their execution times when applied to a broad range of modal specifications and workflow nets of growing size and complexity. Second, it introduces a dedicated benchmark-generation toolchain that has been specially implemented to support and fully automate the process of the proposed experimental protocol. Section 6 details the obtained experimental results and reports on the lessons learned, which constitute the main contribution of this paper. Finally, after discussing related work in Sect. 7, Sect. 8 concludes the paper by underlining the major feedback and suggesting directions for future work.

2 Preliminaries

This section presents preliminaries and basics required for the presentation of the modal specification verification approach considered in this paper. More specifically, it presents Petri net [27], workflow nets [35] and related notions that are used throughout this paper to describe the verification approach.

2.1 Petri Nets

Petri nets, also known as place/transition nets, are a basic model of parallel and distributed systems proposed by Carl Adam Petri [27]. They allow modelling of discrete event systems exhibiting behaviours such as concurrency, conflict, and causal dependency between events in a readable graphical and/or a formal manner. They are widely used to model concurrent processes in theoretical computer science. They are also used to describe chemical reactions, manufacturing processes, supply chains, and so on. Fortunately, for this expressive formal model, several important verification problems, like *reachability problem*, are known to be decidable [24].

Definition 1 (Petri net). A Petri net is a tuple $\langle P, T, F\rangle$ where:

- P is a finite set of places,
- T is a finite set of transitions ($P \cap T = \emptyset$),
- $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.

Let $g \in P \cup T$ and $G \subseteq P \cup T$. We use the following notations:

- $g^\bullet = \{g' | (g, g') \in F\}$, ${}^\bullet g = \{g' | (g', g) \in F\}$,
- $G^\bullet = \cup_{g \in G}\ g^\bullet$, and ${}^\bullet G = \cup_{g \in G}\ {}^\bullet g$.

This definition allows characterizing important structural features such as siphons and traps.

Definition 2 (Siphon). Let $N \subseteq P$ such that $N \neq \emptyset$:

- N is a trap if and only if $N^\bullet \subseteq\ {}^\bullet N$, and
- N is a siphon if and only if ${}^\bullet N \subseteq N^\bullet$.

The *marking* of a Petri net, representing the number of tokens on each place, is a function $M : P \to \mathbb{N}$. It evolves during its execution since transitions change the marking of a Petri net according to the following *firing rules*. A transition t is *enabled* in a marking M if and only if $\forall p \in {}^\bullet t, M(p) \geq 1$. When an *enabled* transition t is *fired*, it *consumes* one token from each place of ${}^\bullet t$ and *produces* one token for each place of $t^\bullet$.

Let M_a and M_b be two markings, and t a transition of a Petri net N, $M_a \xrightarrow{t} M_b$ denotes that the transition t is *enabled* in marking M_a, and *firing* it results in the marking M_b. M_b is then called *directly reachable* from M_a by transition t.

Let $M_1, M_2, .., M_n$ be markings, and $\sigma = t_1, t_2, .., t_{n-1}$ a sequence of transitions of a Petri net N, $M_1 \xrightarrow{\sigma} M_n$ denotes that $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} .. \xrightarrow{t_{n-1}} M_n$. The marking M_n is then said to be *reachable* from M_1 by the sequence of transitions σ. We denote $\mathcal{R}^N(M)$ the set of markings of N *reachable* from a marking M.

With respect to these rules, a transition t is *dead* at marking M if it is not *enabled* in any marking M' *reachable* from M. A transition t is *live* if it is not *dead* in any marking *reachable* from the initial marking. A Petri net system is *live* if each of these transitions is *live*.

In the context of Petri nets, the need for considering classes of Petri nets expressive enough with respect to their modelling capabilities has led to the definition of several subclasses of Petri nets based on structural features. This paper deals with the following well-known and popular Petri net classes:

- State-Machines (SM) without concurrency, but with possible conflicts among tasks (transitions):
$$\forall t \in T, |\ t^\bullet\ | = |^\bullet\ t\ | = 1$$
- Marked-Graphs (MG) without conflict, but there can be concurrent tasks:
$$\forall p \in P, |\ p^\bullet\ | = |^\bullet\ p\ | = 1$$
- Free-Choice nets (FC) where there can be both concurrency and conflict, but not at the same time:
$$\forall p \in P, (|\ p^\bullet\ | \leq 1) \vee ({}^\bullet(p^\bullet) = \{p\})$$

2.2 Workflow Nets

Workflow nets (WF-nets) [35] constitute a special case of Petri nets which are usually used to model the control-flow dimension of a workflow. They allow the modelling of complex workflows exhibiting concurrencies, conflicts, as well as causal dependencies of activities. The use of Petri net-based modelling of workflow systems has been extensively studied [33,35,29], in order to be put into practice. When modelling workflows using workflow nets, the different activities are modelled by transitions, while causal dependencies are modelled by places and arcs. For instance, the Petri net depicted in Fig. 2.1 is a workflow net. The next definitions formally introduce WF-nets.

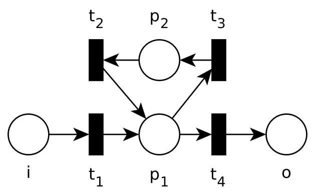


Figure 2.1. An example of a WF-net (E_1)

Definition 3 (Workflow net [35]). A Petri net $N = \langle P, T, F\rangle$ is a workflow net (WF-net) if and only if N is a Petri net, N has two special places i and o, where ${}^\bullet i = \emptyset$ and $o^\bullet = \emptyset$, and for each node $n \in (P \cup T)$ there exists a path from i to o passing through n.

We denote M_i the initial marking (i.e. $M_i(n) = 1$ if $n = i$, and 0 otherwise) and M_o the final marking (i.e. $M_o(n) = 1$ if $n = o$, and 0 otherwise). A *correct* execution of a WF-net is a transition sequence σ such that $M_i \xrightarrow{\sigma} M_o$.

The behaviour of a WF-net is defined as the set Σ of all its correct executions. Given a transition t and an execution σ, the function $O_t(\sigma)$ gives the number of occurrences of t in σ.

In the context of workflow nets, a well-established correctness feature that all workflows should verify is called *soundness* [35]. It states that beside structural properties given by the definition of workflow nets, a *sound* workflow net describes a procedure that will terminate possibly (option to complete), and that when it does there is a token in place o, and all of the other places are empty (proper completion). Additionally, a *sound* workflow net may not contain dead transitions. This correctness criterion constitutes an underlying property of workflow nets that has to be verified in order to ensure correct executions. It has been extensively studied [34, 37], especially in the context of stepwise refinement approach [38], and efficient verification tools have been developed [36, 15, 5].

Definition 4 (Soundness [35]). Let $N = \langle P, T, F \rangle$ be a workflow net, N is sound if and only if:

- $\forall M \in \mathcal{R}^N(M_i), M_o \in \mathcal{R}^N(M)$ (option to complete),
- $\forall M \in \mathcal{R}^N(M_i), (M(o) > 0) \Rightarrow (M = M_o)$ (proper completion), and
- $\forall t \in T, \exists M, M' \in \mathcal{R}^N(M_i), M \xrightarrow{t} M'$ (no dead transitions).

Note that if we assume fairness – transitions that are enabled infinitely often will fire eventually – then the first requirement implies that eventually the final marking is reached. As argued in [35], the fairness assumption is reasonable in the context of workflow management. Indeed, all choices are made implicitly or explicitly by applications, humans or external actors which should not introduce infinite loops.

Needed preliminaries about workflow nets being fixed, the next section introduces the approach, based on constraint systems, to the verification of modal specifications considered in this paper.

3 Constraint-Based Verification of Modal Specifications

In a first time, this section describes modal specifications, more precisely, it presents modal specifications that allow specifiers to describe *necessary* or just *admissible* behaviours as first introduced in [3]. In a second time, this section presents and describes the verification framework based on constraint systems, also introduced in [3], that enables the verification of such modal specifications.

3.1 Modal Specifications

Modal specifications, as presented and used in this paper, are especially useful during the development of workflows systems developed in a top-down fashion, when the original workflows are stepwise refined, each step bringing them closer to the underlying implementations ready for release and production [32].

In the context of stepwise refinement of workflow nets, one of the main principles is that if the initial abstract specification is correct, and the refinement steps preserve correctness, then the resulting specification will be correct by construction. This drives the need to validate some behavioural properties possibly at the early stage of development life cycle.

To this end, modal specifications have been designed to allow *loose* specifications to be expressed by imposing restrictions on transitions. They permit specifiers to indicate that a transition is *necessary* or just *admissible*.

In their simplest form, the modal specifications allow modeller to specify that a transition is a *may*-transition, i.e. a transition which may be enabled, or that a transition is a *must*-transition, i.e. a transition which must be enabled. Verifying that a given transition modality is satisfied by the developed workflow net, or determining the modality of a given transition, has initially motivated the theoretical work in [3, 4].

Nonetheless, as argued in [3, 4] on the case study requirement base, while basic modal specifications are useful, they usually lack expressiveness for real-life applications, as only individual transitions are concerned with. To fill this gap, these papers notably introduce (extended) modal specifications to express requirements on several transitions and on their causalities. Such modal specifications can be used to specify more complex modal and logic links among multiple activities. For example, let t_a and t_b be two activities of the considered process (i.e. two transitions of the workflow net which models the considered process), using (extended) modal specifications, one can express the following modal properties: activity t_a implies activity t_b (causal dependence), activity t_a is completed if and only if activity t_b is also completed (causal co-dependence), either activity t_a or activity t_b is completed but not both (mutual exclusion).

More precisely, in [3], modal specifications allow specifiers to express requirements on several transitions and on their causalities. The modal specifications of a workflow net $N = \langle P, T, F \rangle$ are specified using the language S of well-formed *modal* specification formulae inductively defined by: $\forall t \in T, t$ is a well-formed *modal* formula, and given $A_1, A_2 \in S$, $A_1 \wedge A_2$, $A_1 \vee A_2$, and $\neg A_1$ are well-formed *modal* formulae. These formulae allow specifiers to express modal properties about WF-nets correct executions.

Let $\sigma \in \Sigma$ be a correct execution of N and m a well-formed modal specification formula, we denote $\sigma \models m$ the fact that the modal property expressed by m is satisfied by the execution σ. Formally, given $t \in T$ and $A_1, A_2 \in S(N)$, we have $\sigma \models t \Leftrightarrow O_t(\sigma) > 0$, $\sigma \models (A_1 \wedge A_2) \Leftrightarrow \sigma \models A_1 \wedge \sigma \models A_2$, $\sigma \models (A_1 \vee A_2) \Leftrightarrow \sigma \models A_1 \vee \sigma \models A_2$, and $\sigma \models (\neg A_1) \Leftrightarrow \neg(\sigma \models A_1)$.

Any modal specification formula $m \in S$ can be interpreted as a *may*-formula or a *must*-formula. On the one hand, a *may*-formula describes a admissible behaviour that has to be ensured by at least one correct execution of the WF-net. On the other hand, a *must*-formula describes a necessary behaviour that has to be ensured by all the correct executions of the WF-net.

Modal specifications can thus express a wide range of safety and liveness properties. Typically, a safety property is a property asserting that something bad never happens, whereas a liveness property asserts that something good eventually happens. Therefore a *may*-formula which is expected not to hold can express a safety property. Similarly, a *must*-formula that is expected to hold can be viewed as a liveness property.

Further, given a well-formed *may*-formula (resp. *must*-formula) $m \in S$, a WF-net N satisfies m, written $N \models_{may} m$ (resp. $N \models_{must} m$), when at least one (resp. all) correct execution(s) of N satisfies (resp. satisfy) m. Formally, given $m \in S(N)$:

$$N \models_{may} m \Leftrightarrow \exists\ \sigma \in \Sigma, \sigma \models m, \text{ and}$$

$$N \models_{must} m \Leftrightarrow \forall\ \sigma \in \Sigma, \sigma \models m \wedge N \models_{may} m.$$

For example, regarding the workflow net E_1 depicted in Fig. 2.1, we have $E_1 \models_{may} t_2 \wedge t_3$ as well as $E_1 \models_{must} (\neg t_3) \vee t_4$.

It should be noticed that, according to this definition, the semantic of modal specifications is only defined with respect to the set of correct executions (i.e. Σ) and therefore does not consider all behaviours exhibited by the considered workflows. For instance, executions leading, from the initial marking, to a deadlock marking (i.e. a marking with no enabled transition) different from the final marking are not taken into account. In fact, it is here assumed, based on common practice, that the soundness of the considered workflows are already positively determined as a preprocessing step to the verification of modal specifications.

Further, note that, in contrast to modal specification languages over labelled transition systems [22,14] and Petri nets [13], the considered modal specification language does not specify the behaviours that can, or have to, appear in the subsequent refinements. Indeed, similarly to specification logics such as CTL [9], the modal specification language previously presented and considered in this paper specifies properties of the behaviours appearing at the current refinement stage. In the context of stepwise refinement of workflow nets, this implies that either subsequent refinements must ensure the preservation of the previously verified modal specifications, or that the modal specifications impacted by refinement patterns must be refined accordingly [1].

As the matter of fact, modal specifications are a proper subset of CTL. More precisely, *must*-formula can be expressed as CTL property thanks to the operator A (i.e. along all paths) whereas *may*-formula can be expressed as CTL property thanks to the operator E (i.e. along at least one path). Further details are provided in Appendix A.

Additionally, in order to highlight the numerous use-cases of modal specifications, let us note that properties expressed by modal specification are closely related to properties expressed by deontic logic – i.e. a field of philosophical logic that is concerned with obligation, permission, and related concepts. More precisely, *must*-formulae express obligations whereas *may*-formulae express permission. Deontic logic are largely used throughout the specification of workflows processes in the domain of banking and legal process compliance [16,17].

Finally, let us underline that, when using modal specifications, one cannot express temporal aspects (i.e. activity t_a must be performed before activity t_b). Indeed, modal specifications aim at expressing modal properties that can be efficiently verified by leveraging abstractions of the systems under analysis. Such abstractions (e.g., the abstractions presented in the next section) present advantages over other verification methodologies (e.g., model checking), notably thanks to their restricted use-cases (e.g., abstraction of temporal aspects). Therefore, while not as expressive as other previously mentioned specification logics, modal specifications are of prime interest when it comes to verify modal specifications that do not involve temporal aspects as they enable the use of a specialised verification method.

3.2 Modal Specifications Verification Method

This section provides an overall detailed description of the verification method introduced in [3] to verify modal specifications of workflow nets. This method is based on the resolution of constraint systems.

Basically, a constraint system is a set of constraints (properties), which must be satisfied by the solution of the problem it models. To achieve that, each variable appearing in a constraint of the system should takes its value from its domain. Such a system defines a Constraint Satisfaction Problem (CSP).

Definition 5 (Constraint Satisfaction Problem). A Constraint Satisfaction Problem, CSP for short, is a tuple $\Omega =< X, D, C >$ where:

- X is a set of variables $\{x_1, \ldots, x_n\}$,
- D is a set of domains $\{d_1, \ldots, d_n\}$ (d_i is the domain associated with the variable x_i),
- C is a set of constraints $\{c_1(X_1), \ldots, c_m(X_m)\}$ (c_j is a constraint on a subset X_j of the variables of X).

A CSP thus models NP-complete problems as search problems where the corresponding search space is the Cartesian product space $d_1 \times \ldots \times d_n$. The solution of a CSP Ω is computed by a labelling function $\mathcal{L}$, which provides a set ν (called valuation function) of tuples assigning each variable x_i of X to one value from its domain

d_i such that all the constraints C are satisfied. More formally, ν is consistent—or satisfies a constraint $c(X)$ of C—if the projection of ν on X is in $c(X)$. If ν satisfies all the constraints of C, denoted by $\Omega \models \nu$, then Ω is a consistent or satisfiable CSP.

In our context, to verify a modal specification m of a WF-net N, the constraint system is composed of a set of constraints representing the correct executions of N completed with the constraint issued from m. This constraint system can then be solved to validate or invalidate the modal specification m regarding the WF-net N. Considering a WF-net $N = (P, T, F)$, the method starts by modelling all the correct executions leading from M_a to M_b, i.e. all σ such that $M_a \xrightarrow{\sigma} M_b$ through their decomposition into segments structurally verifiable. To reach that, the following constraint systems are defined:

Definition 6 (State Equation Constraint System). Let $N = \langle P, T, F\rangle$ be a workflow net, and M_a, M_b two markings of N, the state equation constraint system $\mathcal{S}(N, M_a, M_b)$, associated with it, is:

– $\forall p \in P,\ \nu(p) = \sum_{t \in p^\bullet} \nu(t) + M_b(p) = \sum_{t \in {}^\bullet p} \nu(t) + M_a(p)$

where $\nu : P \cup T \to \mathbb{N}$ is a valuation function.

The solutions of the constraint system of Definition 6 are related to the execution of N, as stated in Theorem 1 issued from [26].

Theorem 1 (Reachability Necessary Condition). *Let $N = \langle P, T, F\rangle$ be a workflow net. If $M_a \xrightarrow{*} M_b$ then there exists a valuation satisfying $\mathcal{S}(N, M_a, M_b)$.*

More precisely, the constraint system $\mathcal{S}(N, M_a, M_b)$ models the fact that for each place p of N, the number of token(s) entering p plus the number of token(s) in $M_a(p)$ is equal to the number of tokens leaving p plus the number of token(s) in $M_b(p)$. Let ν be a valuation function satisfying $\mathcal{S}(N, M_a, M_b)$, ν aims at modelling a transition sequence σ such that $\forall t \in T, O_t(\sigma) = \nu(t)$, denoted $\nu \models \sigma$. However, there is no guarantee that there exists such σ corresponding to an execution of N reaching the marking M_b from the marking M_a. Solutions of $\mathcal{S}(N, M_a, M_b)$ which do not correspond to an execution of N are called spurious solutions. Indeed, spurious solutions can appear because the order of transition firing is not taken into account in the modelled execution. For example, consider E_1 the workflow net depicted in Fig. 2.1, the valuation function $\nu_1 : P \cup T \to \mathbb{N}$ defined such that $\forall\ n \in \{i, p_1, p_2, t_2, t_3\}, \nu_1(n) = 1$ and $\forall\ n \in \{t_1, t_4, o\}, \nu_1(n) = 0$ satisfies $\mathcal{S}(E_1, M_i, M_i)$ but does not model an execution of E_1. Indeed, in this example, the transitions involved (i.e. t_2 and t_3) can not be fired in any order from the initial marking (i.e. M_i).

Nevertheless, as stated in Theorem 1, if there is no valuation satisfying $\mathcal{S}(N, M_a, M_b)$ then the marking M_b is not reachable from marking M_a. In this sense the constraint system $\mathcal{S}(N, M_a, M_b)$ defines an over-approximation of all the executions of N reaching the marking M_b from the marking M_a.

In order to dismiss the spurious solutions of $\mathcal{S}$, [3] proposes to refine it. To do so, it considers structural features of the subnets corresponding to executions modelled by $\mathcal{S}$.
Let ν be a solution of the $\mathcal{S}(N, M_a, M_b)$ constraint system, the subnet associated with it is an ordinary Petri net such that only the places and transitions of N involved in the ν valuation are considered. Note that the decomposition process also removes the places which have a greater or equal number of tokens in the marking M_a (resp. M_b) with respect to the number of tokens consumed from (resp. produced in) those places.

Definition 7 (State Equation Solution Subnet). Let $N = \langle P, T, F\rangle$ be a workflow net, M_a, M_b two markings of N, $\nu : P \cup T \to \mathbb{N}$ a satisfying valuation of $\mathcal{S}(N, M_a, M_b)$, $\mathcal{U}^+ : P \mapsto \mathbb{N}$ a function such that $\forall p \in P$, $\mathcal{U}^+(p) = \sum_{t \in p^\bullet} \nu(t)$ and $\mathcal{U}^- : P \mapsto \mathbb{N}$ a function such that $\forall p \in P$, $\mathcal{U}^-(p) = \sum_{t \in {}^\bullet p} \nu(t)$. We define the subnet $sN(\nu) = \langle sP, sT, sF\rangle$ as an ordinary Petri net where:

– $sP = \{p \in P \mid \nu(p) > 0 \wedge (\mathcal{U}^+(p) > M_a(p) \vee \mathcal{U}^-(p) > M_b(p))\}$
– $sT = \{t \in T \mid \nu(t) > 0\}$, and
– $sF = \{(a, b) \in F \mid a \in (sP \cup sT) \wedge b \in (sP \cup sT)\}$

Let us now recall that an unmarked siphon cannot be marked and a marked trap cannot be unmarked[26]. Therefore, the subnet $sN(\nu)$ of a non spurious solution ν of $\mathcal{S}(N, M_a, M_b)$ does not contain a trap nor a siphon. Indeed, as a marked trap cannot be unmarked and places marked in the final marking M_b have been removed, $sN(\nu)$ cannot contain a trap. Likewise as an unmarked siphon cannot be marked and places of $sN(\nu)$ must have at least a token consumed, $sN(\nu)$ cannot contain a siphon.

To illustrate this, let us consider the subnet associated with the valuation ν_1 previously defined as an example of spurious solution of $\mathcal{S}(E_1, M_i, M_i)$. This subnet, depicted in Fig. 3.1, contains a siphon composed of the places p_1 and p_2.

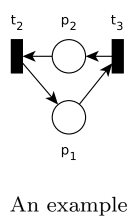


Figure 3.1. An example of a WF-net

We continue the reasoning by noting an interesting property of the subnet $sN(\nu)$ of any solution ν of $\mathcal{S}(N, M_a, M_b)$ which relates the presence of siphons to the presence of traps, and vice-versa, as stated in Theorem 2 issued from [3].

Theorem 2 (Siphon/Trap Property of Subnets). *Let $N = \langle P, T, F\rangle$ be a workflow net, M_a, M_b two markings of N, $\nu : P \times T \to \mathbb{N}$ a valuation satisfying $\mathcal{S}(N, M_a, M_b)$ and $sN(\nu)$ the subnet associated with ν.*

If $sN(\nu)$ contains a trap (resp. siphon) G then G is also a siphon (resp. trap).

Therefore, according to Theorem 2, one can decide the presence of siphons and traps in a subnet by deciding the presence of siphons only (or, equivalently, the presence of traps only). The presence of siphons in an ordinary Petri net can be decided by evaluating the satisfiability of the constraint system introduced in Theorem 3 that is issued from [3]

Theorem 3 (Siphon Presence Constraint System). *Let $N = \langle P, T, F\rangle$ be an ordinary Petri net and $\mathcal{B}(N)$ the constraint system defined as follows:*

– $\forall p \in P, \forall t \in {}^\bullet p. \sum_{p' \in {}^\bullet t} \xi(p') \geq \xi(p)$
– $\sum_{p \in P} \xi(p) > 0$

where $\xi : P \to \{0, 1\}$ is a valuation function.

N contains a siphon if and only if there is a valuation satisfying $\mathcal{B}(N)$.

Using the constraint system $\mathcal{S}$ together with the constraint system $\mathcal{B}$ leads to the definition of the constraint system $\mathcal{Q}$ as follows.

Definition 8 (State Equation + Absence of Siphon Constraint System). Let $N = \langle P, T, F\rangle$ be a workflow net and M_a, M_b two markings of N, the constraint system $\mathcal{Q}(N, M_a, M_b)$, associated with it, is:

– $\mathcal{S}(N, M_a, M_b) \models \nu$
– $\nexists\ \xi$ such that $\mathcal{B}(sN(\nu)) \models \xi$

where $\nu : P \cup T \to \mathbb{N}$ is a valuation function.

While many of the spurious solutions of $\mathcal{S}(N, M_a, M_b)$ are discarded from the solutions of $\mathcal{Q}(N, M_a, M_b)$, the latter constraint system still defines an over-approximation of the valid executions of N leading to M_b from M_a. This is because the absence of traps and siphons in the subnets of solutions of $\mathcal{Q}(N, M_a, M_b)$ is only a necessary but not a sufficient condition to the existence of a valid execution of N leading to M_b from M_a.

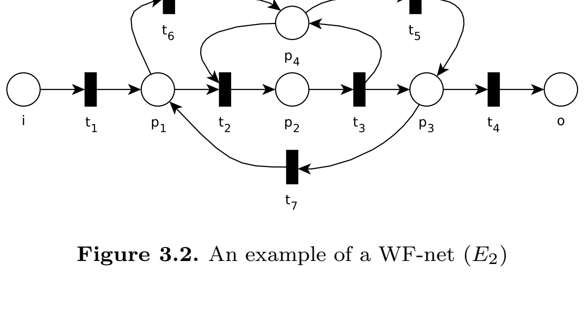


Figure 3.2. An example of a WF-net (E_2)

To illustrate this claim, let us consider the E_2 workflow depicted in Fig. 3.2. The valuation ν_2, where $\forall\ n \in \{i, p_2, o, t_1\ , t_2, t_3, t_4, t_5, t_6, t_7\}$, one has $\nu_2(n) = 1$ and $\forall\ n \in \{p_1, p_3, p_4\}$, one has $\nu_2(n) = 2$, is a valuation satisfying $\mathcal{Q}(E_2, M_i, M_o)$. However, the transition t_2 of this workflow E_2 is a dead transition from the initial marking M_i. Indeed, to be enabled, the transition t_2 requires the presence of two tokens (one in place p_1 and one in place p_4), but the initial marking only has one token and the only transition creating a token is t_3 which requires a token to be produced by t_2 in place p_2. Therefore there exists no execution σ such that $M_i \xrightarrow{\sigma} M_o$ and $O_{t_2}(\sigma) > 0$. It follows that ν_2 is a spurious solution of $\mathcal{Q}(E_2, M_i, M_o)$.

While defining an over-approximation might be useful for the verification of safety properties, in our case, our goal is to be able to model any executions of a workflow net with enough precision in order to verify modal specifications. To further refine $\mathcal{Q}$ and to define sufficient conditions structurally, additional constraints are required. To this end, [3] considers solution ν of $\mathcal{Q}(N, M_a, M_b)$ such that $sN(\nu)$ is a marked graph. The resulting constraint system is denoted $\mathcal{D}$.

Definition 9 (State Equation + Absence of Siphon + MG Constraint System). Let $N = \langle P, T, F\rangle$ be a workflow net, and M_a, M_b two markings of N, the constraint system $\mathcal{D}(N, M_a, M_b)$ associated with it, is:

– $\forall p \in P,\ \sum_{t \in {}^\bullet p} \nu(t) \leq 1 \wedge \sum_{t \in p^\bullet} \nu(t) \leq 1$
– $\mathcal{Q}(N, M_a, M_b) \models \nu$

where $\nu : P \cup T \to \mathbb{N}$ is a valuation function.

The solutions of the constraint system of Definition 9 are related to the execution of a workflow net N, as stated in Theorem 4 issued from [3].

Theorem 4 (Path Existence). *Let $N = \langle P, T, F\rangle$ be a workflow net and M_a, M_b two markings of N. If there exists a valuation $\nu : P \cup T \to \mathbb{N}$ such that $\mathcal{D}(N, M_a, M_b) \models \nu$ then there exists a transition sequence σ such that $M_a \xrightarrow{\sigma} M_b$ and $\nu \models \sigma$.*

As stated in Theorem 4, any solutions of $\mathcal{D}(N, M_a, M_b)$ model at least one valid execution of N leading to M_b from M_a. Note the solutions of $\mathcal{D}$ are abstractions of the executions they model as the ordering of the transitions they involve is not explicitly computed. However, as N might not belong to the subclass of marked graph, not all valid executions of N leading to M_b from M_a can be modelled by solutions of $\mathcal{D}(N, M_a, M_b)$. In this sense, the solutions of $\mathcal{D}(N, M_a, M_b)$ define an under-approximation of the valid executions of N leading to M_b from M_a.

Every execution modelled by the constraint system $\mathcal{D}$ is called a *segment*, and [3] proceeds by decomposing any execution of a workflow net into such segments. The resulting constraint system is denoted $\mathcal{U}$.

Definition 10 (k-segment Execution Constraint System). Let $N = \langle P, T, F \rangle$ be a workflow net, M_a, M_b two markings of N, and $k \in \mathbb{N}$ a number of segments, the constraint system $\mathcal{U}(N, M_a, M_b, k)$, associated with it, is:

– $\exists\ M_1, \nu_1, \mathcal{D}(N, M_a, M_1) \models \nu_1$
– $\forall\ i \in \{2, .., k-1\}, \exists\ M_i, \nu_i, \mathcal{D}(N, M_{i-1}, M_i) \models \nu_i$
– $\exists\ \nu_k, \mathcal{D}(N, M_{k-1}, M_b) \models \nu_k$
– $\forall\ n \in P \cup T, \nu(n) = \sum_{i \in \{1,..,k\}} \nu_i(n)$

where $\nu : P \cup T \to \mathbb{N}$ is a valuation function.

The solutions of the constraint system of Definition 10 are related to the execution of a workflow net N, as stated in Theorem 5 issued from [3].

Theorem 5 (Path Existence). *Let $N = \langle P, T, F \rangle$ be a workflow net and M_a, M_b two markings of N. There exists a transition sequence σ such that $M_a \xrightarrow{\sigma} M_b$ if and only if there exist $k \in \mathbb{N}$ a number of segments and $\nu : P \cup T \to \mathbb{N}$ a valuation function such that $\mathcal{U}(N, M_a, M_b, k) \models \nu$ and $\nu \models \sigma$.*

By Theorem 5, every execution composed of at most k segments and leading to a marking M_b from a marking M_a of a workflow net N can be modelled by the constraint system $\mathcal{U}(N, M_a, M_b, k)$.

We now have recalled how ordinary workflow nets correct execution can by modelled by constraint systems according to previous work in [3]. More precisely, we have reviewed how every execution could be modelled by segments structurally verifiable. We now continue by describing how such modelling can be used to verify modal specifications over ordinary workflow nets.

Intuitively, a workflow net N is valid with respect to a *may*-formula m if and only if there exists a correct execution σ of N such that the modal property expressed by m is satisfied by the execution σ (i.e. $N \models_{may} m \Leftrightarrow \exists\ \sigma \in \Sigma, \sigma \models m$). Likewise a workflow net N is valid with respect to a *must*-formula m if and only if it is a valid *may*-formula ans there does not exist a correct execution σ of N such that the modal property expressed by $\neg m$ is satisfied by the execution σ (i.e. $N \models_{must} m \Leftrightarrow \nexists\ \sigma \in \Sigma, \sigma \models (\neg m)$).

Furthermore, when determining whether or not a workflow net satisfies the modal properties of interest, two decision problems are distinguished. The first one, called the *K-bounded validity of a modal formula*, only considers executions formed by K segments, at most. The second one, called the *unbounded validity of a modal formula*, deals with executions formed by an arbitrary number of segments; it generalises the first problem.

In order to verify modal specifications, the method proposed in [3] relies on their expression by constraints. To build these constraints, for every transition $t \in T$, the corresponding terminal symbol of the modal formulae is replaced by $\nu(t) > 0$, where ν is the valuation of the constraint system.

Given a modal formula $f \in S$, $C(f, \nu)$ is the constraint built from f, where ν is a valuation of the constraint system. Upon the expression of a modal property by a constraint system, we build a constraint system extending the constraint system $\mathcal{U}(N, M_i, M_o, k)$ which models the correct execution of N composed of at most k segments.

Definition 11 (k-segment Modal Execution Constraint System). Let $N = \langle P, T, F \rangle$ be a workflow net, $k \in \mathbb{N}$ a number of segments, and $m \in S(N)$ a modal property, the associated constraint system $\mathcal{V}(N, k, m)$ is defined as follow:

– $\mathcal{U}(N, M_i, M_o, k) \models \nu$
– $\mathcal{C}(N, m) \models \nu$

where $\nu : P \cup T \to \mathbb{N}$ is a valuation function.

The solutions of the constraint system derived from Definition 11 are related to the execution of a workflow net N, as stated in Theorem 6 issued from [3].

Theorem 6 (Path Existence). *Let $N = \langle P, T, F \rangle$ be a workflow net and $m \in S(N)$ a modal property.*

There exists a transition sequence σ such that $M_i \xrightarrow{\sigma} M_o$ and $\sigma \models m$ if and only if there exists $k \in \mathbb{N}$ and $\nu : P \cup T \to \mathbb{N}$ a valuation function such that $\mathcal{V}(N, k, m) \models \nu$, and $\nu \models \sigma$.

By Theorem 6, there exists a correct execution of N composed of at most k segments satisfying a modal property $m \in S(N)$ if and only if there exists a valuation satisfying the constraint system $\mathcal{V}(N, k, m)$. It follows that we can determine the *K-bounded validity* of a modal property m, by determining the satisfiability of $\mathcal{V}(N, K, m)$. Consequently let m be a *may*-formula (resp. a *must*-formula), the workflow net N is said to be K-bounded valid with respect to m if and only if $\exists\ \nu, \mathcal{V}(N, K, m) \models \nu$ (resp. $\nexists\ \nu_1, \mathcal{V}(N, K, \neg m) \models \nu_1 \wedge \exists\ \nu_2, \mathcal{V}(N, K, m) \models \nu_2$).

It follows that using a fixed K, the *K-bounded validity of a modal formula*, i.e. the validity of the modal formula over correct executions formed by at most K segments, can be inferred by evaluating the satisfiability of the corresponding constraint system. Furthermore, it has been shown that for K sufficiently large the *K-bounded validity of a modal formula* corresponds to its *unbounded validity* [3].

To summarize, this method proposes to encompass both the modelling of workflow nets executions through their decomposition into structurally verifiable segments and the modal specification within the framework of constraints modelling. It enables their verification thanks to the use of mature and efficient constraint solving tools. The next section introduces the toolchain developed to support this method.

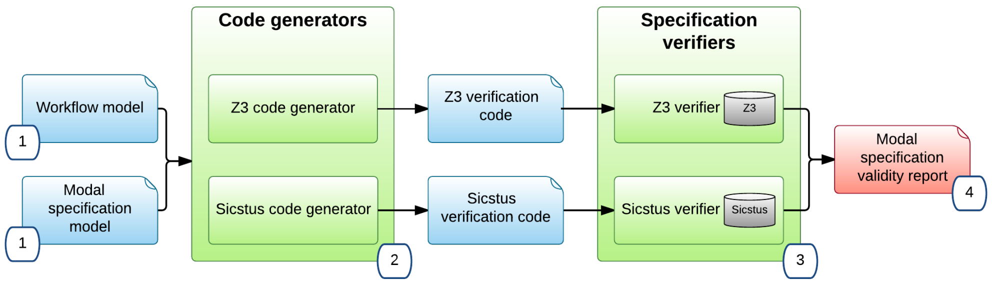


Figure 4.1. Modal specifications verification toolchain

4 Modal Specifications Verification Tool

This section describes the toolchain developed to experimentally validate and evaluate the modal specification verification method described in Sect. 3.2. It presents a complete and mature toolchain able to carry out the modal specification verification of workflow nets based on constraints system solving. Figure 4.1 depicts the architecture of the verification toolchain, which enables to verify modal specifications using CLP or SMT solving.

First, the workflow net and the modal specification models are given as inputs to the verification software(1). Both models conform to an ad-hoc standard, i.e. a dedicated meta-model, so that all information needed by the verification tool is provided.

Afterwards, the two code generators use the information provided by the workflow and the modal specification(1) to generate the code corresponding to the constraint system defined in Sect. 3.2. Such codes are expressed in the input format of the targeted constraint solver(2). To evaluate the produced constraint systems, this toolchain relies on either SMT-Lib for Z3 [10] version 4.4.0, an SMT solver that finished first during the 2014 SMT-COMP challenge[1] for solving non-linear arithmetic problems, or SICStus Prolog [8] version 4.3.2, a CLP solver (embedding a CLP(FD) library) that obtained the third place during the 2014 MiniZinc challenge[2].

Next, by applying the verification algorithm described in Sect. 3.2, the verification module queries the related solver with the generated code to determine the validity of the modal specification(3).

Finally, a report is produced(4), giving the verdict about the validity of the modal specification as well as the verification time.

Figures 4.2 and 4.4 illustrate the encoding used to model the constraints seen in Sect. 3.2 respectively for the SMT-Lib and Prolog language. The constraints are generated for the workflow E_1 depicted by Fig. 2.1 and the modal formula to be checked is $E_1 \models_{may} t_2 \wedge t_3$, asking whether there exists a valid execution of E_1 such that both transitions t_2 and t_3 are fired. Note that this is indeed the case since for instance the path $\sigma = t_1, t_3, t_2, t_3, t_2, t_4$ is valid and fires both t_2 and t_3 at least once.

[1] www.smtcomp.org

[2] www.minizinc.org/challenge.html

We use a particular naming convention to encode each valuation and constraint seen in Sect. 3.2. Note that this convention is followed for both Prolog and SMT-Lib encodings. Let P and T be respectively the set of places and transitions in E_1. For all $p \in P$, all $t \in T$ and all $i \in \mathbb{N}$:

– A variable Ap stands for the marking of the place p before firing a sequence of transitions (i.e. Ap $\equiv M_a(p)$).
– A variable Bp stands for the marking of the place p after firing a sequence of transitions (i.e. Bp $\equiv M_b(p)$).
– A variable Pp stands for the valuation of the place p after firing a sequence of transitions (i.e. Pp $\equiv \nu(p)$).
– A variable Tt stands for the valuation of the transition t (i.e. Tt $\equiv \nu(t)$).
– A variable Xp stands for the valuation of the place p in the subnet used for the detection of siphons (i.e. Xp $\equiv \xi(p)$).
– The variables M1p, M2p and M3p respectively stand for $M_1(p)$, $M_2(p)$ and $M_3(p)$.

Each constraint seen in Sect. 3.2 has also the corresponding function in the encoding.

The functions `initialMarking` and `finalMarking` respectively encode the initial marking and the final marking of a workflow.

They restrict the modelled executions of a workflow net to its correct executions. Relaxing these constraints would enable the proposed method to be applicable to other classes of Petri nets. For example, by removing the constraint associated with the final marking, it would be possible for the proposed approach to model executions of Petri nets denoting reactive behaviour, without any final place.

The function `stateEquation` encodes the state equation constraint system (see Def. 6), therefore taking M_a (`Ai`, `Ao`, `Ap1` and `Ap2`) and M_b (`Bi`, `Bo`, `Bp1` and `Bp2`) as parameters and *returning* a valuation for each place (`Pi`, `Po`, `Pp1` and `Pp2`) and each transition (`Tt1`, `Tt2`, `Tt3` and `Tt4`) of the workflow. The function `stateEquation` first ensures that all variables belong to $\mathbb{N}$ (see for instance the constraint `Tt3 >= 0`) and then applies the state equation constraint to each place of the workflow (see for instance for the place p_1 the constraints `Pp1 = Tt3 + Tt4 + Bp1` and `Pp1 = Tt1 + Tt2 + Ap1`).

```
; Variables definitions here
; (declare-fun M1i () Int)
; ...
; (declare-fun T2t4 () Int)

(define-fun initialMarking ((Ai Int) (Ao Int) (Ap1 Int) (Ap2 Int)) Bool
    (and (= Ai 1) (= Ao 0) (= Ap1 0) (= Ap2 0))
)

(define-fun finalMarking ((Bi Int) (Bo Int) (Bp1 Int) (Bp2 Int)) Bool
    (and (= Bo 1) (= Bi 0) (= Bp1 0) (= Bp2 0))
)

(define-fun stateEquation
    (
        (Ai Int) (Ao Int) (Ap1 Int) (Ap2 Int)
        (Bi Int) (Bo Int) (Bp1 Int) (Bp2 Int)
        (Pi Int) (Po Int) (Pp1 Int) (Pp2 Int)
        (Tt1 Int) (Tt2 Int) (Tt3 Int) (Tt4 Int)
    ) Bool
    (and
        (>= Ai 0) (>= Ao 0) (>= Ap1 0) (>= Ap2 0)
        (>= Bi 0) (>= Bo 0) (>= Bp1 0) (>= Bp2 0)
        (>= Pi 0) (>= Po 0) (>= Pp1 0) (>= Pp2 0)
        (>= Tt1 0) (>= Tt2 0) (>= Tt3 0) (>= Tt4 0)
        (= (+ Ai) Pi) (= (+ Bi Tt1) Pi)
        (= (+ Ao Tt4) Po) (= (+ Bo) Po)
        (= (+ Ap1 Tt1 Tt2) Pp1)
        (= (+ Bp1 Tt3 Tt4) Pp1)
        (= (+ Ap2 Tt3) Pp2)
        (= (+ Bp2 Tt2) Pp2))
)

(define-fun formula ((Tt2 Int) (Tt3 Int)) Bool (and (> Tt2 0) (> Tt3 0)))

(define-fun noSiphon
    (
        (Ai Int) (Ao Int) (Ap1 Int) (Ap2 Int)
        (Bi Int) (Bo Int) (Bp1 Int) (Bp2 Int)
        (Pi Int) (Po Int) (Pp1 Int) (Pp2 Int)
        (Tt1 Int) (Tt2 Int) (Tt3 Int) (Tt4 Int)
    ) Bool
    (and (not (exists
        ((Xi Int) (Xo Int) (Xp1 Int) (Xp2 Int))
        (and
            (and
                (> (+ Xi Xo Xp1 Xp2) 0)
                (and (>= Xi 0) (<= Xi 1))
                (and (>= Xo 0) (<= Xo 1))
                (and (>= Xp1 0) (<= Xp1 1))
                (and (>= Xp2 0) (<= Xp2 1))
                (=> (or (> Ai 0) (> Bi 0) (= Pi 0)) (= Xi 0))
                (=> (or (> Ao 0) (> Bo 0) (= Po 0)) (= Xo 0))
                (=> (or (> Ap1 0) (> Bp1 0) (= Pp1 0)) (= Xp1 0))
                (=> (or (> Ap2 0) (> Bp2 0) (= Pp2 0)) (= Xp2 0))
                (=> (> Tt1 0) (and (>= (+ Xi) Xp1)))
                (=> (> Tt2 0) (and (>= (+ Xp2) Xp1)))
                (=> (> Tt3 0) (and (>= (+ Xp1) Xp2)))
                (=> (> Tt4 0) (and (>= (+ Xp1) Xo)))
            )
        )
    )))
)

(define-fun markedGraph
    (
        (Tt1 Int) (Tt2 Int) (Tt3 Int) (Tt4 Int)
    ) Bool
    (and
        (<= T1 1)
        (<= (+ T1 T2) 1) (<= (+ T3 T4) 1)
        (<= T3 1) (<= T2 1)
        (<= T4 1)
    )
)

(define-fun segment
    (
        (Ai Int) (Ao Int) (Ap1 Int) (Ap2 Int)
        (Bi Int) (Bo Int) (Bp1 Int) (Bp2 Int)
        (Pi Int) (Po Int) (Pp1 Int) (Pp2 Int)
        (Tt1 Int) (Tt2 Int) (Tt3 Int) (Tt4 Int)
    ) Bool
    (and
        (stateEquation
            Ai Ao Ap1 Ap2
            Bi Bo Bp1 Bp2
            Pi Po Pp1 Pp2
            Tt1 Tt2 Tt3 Tt4
        )
        (noSiphon
            Ai Ao Ap1 Ap2
            Bi Bo Bp1 Bp2
            Pi Po Pp1 Pp2
            Tt1 Tt2 Tt3 Tt4
        )
    )
)
```

Figure 4.2. SMT-Lib code of a segment of workflow

```
(assert (initialMarking M1i M1o M1p1 M1p2))
(assert (finalMarking M3i M3o M3p1 M3p2))
(assert (formula (+ T1t2 T2t2) (+ T1t3 T2t3)))
(assert (segment
    M1i M1o M1p1 M1p2 M2i M2o M2p1 M2p2
    P1i P1o P1p1 P1p2 T1t1 T1t2 T1t3 T1t4
))
(assert (markedGraph T1t1 T1t2 T1t3 T1t4))
(assert (segment
    M2i M2o M2p1 M2p2 M3i M3o M3p1 M3p2
    P2i P2o P2p1 P2p2 T2t1 T2t2 T2t3 T2t4
))
(assert (markedGraph T2t1 T2t2 T2t3 T2t4))
(check-sat-using smt)
(get-model)
```

Figure 4.3. Input encoded in SMT-Lib language

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

initialMarking([1, 0, 0, 0]).

finalMarking([0, 1, 0, 0]).

stateEquation(
    [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
    [Pi, Po,Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4]
):-
    domain([Ai, Ao, Ap1, Ap2], 0, sup),
    domain([Bi, Bo, Bp1, Bp2], 0, sup),
    domain([Pi, Po, Pp1, Pp2], 0, 50),
    domain([Tt1, Tt2, Tt3, Tt4], 0, 50),
    sum([Ai], #=, Pi),
    sum([Bi, Tt1], #=, Pi),
    sum([Ao, Tt4], #=, Po),
    sum([Bo], #=, Po),
    sum([Ap1, Tt1, Tt2], #=, Pp1),
    sum([Bp1, Tt3, Tt4], #=, Pp1),
    sum([Ap2, Tt3], #=, Pp2),
    sum([Bp2, Tt2], #=, Pp2).

formula([Tt2, Tt3]):- ((Tt2 #> 0), (Tt3 #> 0)).

buildSubnet(
    [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
    [Pi, Po, Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4],
    [Xi, Xo, Xp1, Xp2]
):-
    domain([Xi, Xo, Xp1, Xp2], 0, 1),
    (((Ai #> 0) #\/ (Bi #> 0) #\/ (Pi #= 0)) #=> (Xi #= 0)),
    (((Ao #> 0) #\/ (Bo #> 0) #\/ (Po #= 0)) #=> (Xo #= 0)),
    (((Ap1 #> 0) #\/ (Bp1 #> 0) #\/ (Pp1 #= 0)) #=> (Xp1 #= 0)),
    (((Ap2 #> 0) #\/ (Bp2 #> 0) #\/ (Pp2 #= 0)) #=> (Xp2 #= 0)).

siphon([Tt1, Tt2, Tt3, Tt4], [Xi, Xo, Xp1, Xp2]):-
    sum([Xi, Xo, Xp1, Xp2], #>, 0),
    ((Tt1 #> 0) #=> (((Xi) #>= Xp1))),
    ((Tt2 #> 0) #=> (((Xp2) #>= Xp1))),
    ((Tt3 #> 0) #=> (((Xp1) #>= Xp2))),
    ((Tt4 #> 0) #=> (((Xp1) #>= Xo))),
    labeling([ffc, step, up], [Xi, Xo, Xp1, Xp2]).

noSiphon(
    [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
    [Pi, Po, Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4]
):-
    buildSubnet(
        [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
        [Pi, Po, Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4],
        [Xi, Xo, Xp1, Xp2]
    ),
    labeling([ffc, step, up], [Tt1, Tt2, Tt3, Tt4]),
    \+ siphon([Tt1, Tt2, Tt3, Tt4], [Xi, Xo, Xp1, Xp2]).

markedGraph(
    [T1, T2, T3, T4]
):-
    T1 #<= 1,
    T1 + T2 #<= 1, T3 + T4 #<= 1,
    T3 #<= 1, T2 #<= 1,
    T4 #<= 1.

segment(
    [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
    [Pi, Po,Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4]
):-
    stateEquation(
        [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
        [Pi, Po,Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4]
    ),
    noSiphon(
        [Ai, Ao, Ap1, Ap2], [Bi, Bo, Bp1, Bp2],
        [Pi, Po,Pp1, Pp2], [Tt1, Tt2, Tt3, Tt4]
    ).
```

Figure 4.4. SICStus code of a segment of workflow

The function `formula` encodes the modal formula to be verified. This function expresses the exact same constraint as the one expressed by the formula (in our case $t_2 \wedge t_3$) and states that the valuation of each transition appearing in it must be greater than 0. Therefore, for the formula $t_2 \wedge t_3$, we have the constraint `Tt2 > 0` $\wedge$ `Tt3 > 0`.

The function `noSiphon` encodes the absence of siphon constraint system (see Def. 8). Note that for SICStus (Fig. 4.4), the functions `buildSubnet` and `siphon` are subsidiary functions respectively used to build the subnet as defined in Def. 7 and to check the presence of a siphon. The `noSiphon` function ensures that there exists no valuation such that $\mathcal{B}(N)$ (see Th. 3) is satisfied.

The function `markedGraph` encodes the marked graph constraint system (see Def. 9). It ensures that the segment proposed by the solvers is indeed a marked graph, meaning that there can only be at most one transition before and one transition after each place.

The function `segment` encodes the path existence constraint system (see Th. 4). If there exists a valuation satisfying this constraint, there exists a valid sequence of transition (also called a segment of execution) leading from the marking M_a to the marking M_b. The expression of the k-segments constraint defined in Def. 10 is implicitly achieved by successive calls to the `segment` function (for more details, see the explanation for the inputs given below).

The encoding in Fig. 4.3 and 4.5 allows us to determine, using respectively Z3 and SICStus, whether there exists a correct execution of the workflow in Fig. 2.1 made of two segments such that both `t2` and `t3` are fired. These inputs implicitly express the k-segment constraint given in Def. 11. Note that for this example, we used a may specification. Thus, if the solvers find a valuation for these inputs, it means that the specification is valid, since there exists a valid sequence of transitions from the initial marking to the final marking with two segments.

```
initialMarking([M1i, M1o, M1p1, M2p2]),
finalMarking([M3i, M3o, M3p1, M3p2]),
segment(
    [M1i, M1o, M1p1, M1p2], [M2i, M2o, M2p1, M2p2],
    [P1i, P1o, P1p1, P1p2], [T1t1, T1t2, T1t3, T1t4]
),
markedGraph([T1t1, T1t2, T1t3, T1t4]),
segment(
    [M2i, M2o, M2p1, M2p2], [M3i, M3o, M3p1, M3p2],
    [P2i, P2o, P2p1, P2p2], [T2t1, T2t2, T2t3, T2t4]
),
markedGraph([T2t1, T2t2, T2t3, T2t4]),
S1 #= T1t2 + T2t2, S2 #= T1t3 + T2t3,
formula([S1, S2]).
```

Figure 4.5. Input encoded for SICStus Prolog

```
M1i=1, M1o=0, M1p1=0, M1p2=0, M2i=0, M2o=0, M2p1=0, M2p2=1,
M3i=0, M3o=1, M3p1=0, M3p2=0
P1i=1, P1o=0, P1p1=1, P1p2=1, P2i=0, P2o=1, P2p1=1, P2p2=1,
T1t1=1, T1t2=0, T1t3=1, T1t4=0, T2t1=0, T2t2=1, T2t3=0, T2t4=1
```

Figure 4.6. Output given by the solvers

These two segments are given in Fig. 4.7 and derived from the valuation proposed by the solvers (see Fig. 4.6), starting (resp. ending) in the initial (resp. final) marking where only the input place `i` (resp. output place `o`) is marked. The first segment brings the token from the source place i to the place p_2 by firing t_1 and t_3 successively. The second segment brings the token from the place p_2 to the sink place o by firing t_2 and t_4.

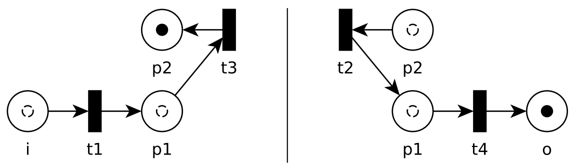


Figure 4.7. The two segments of execution given by both solvers

Let us notice that when using SICStus, multiple labelling heuristics can be used. Figure 4.4 illustrates the use of the *First Fail Cut* strategy (`ffc`, `step` and `up`), given in lines 44 and 55, that is the more efficient CLP heuristic observed during experiments.

5 Experimental Evaluation of Modal Specifications Verification

This section presents the experimental evaluation of the verification of modal specifications over workflow nets accordingly to the approach presented in Sect. 3.2, and thanks to the implemented toolchain described in Sect. 4. To reach this goal, we first detail the objectives of the proposed experimental evaluation. Second, we define the experimental protocol and introduce the toolchain implemented in order to carry it on. Finally, the experimental results obtained according to the defined experimental protocol are presented and discussed.

5.1 Objectives

The primary objectives of this experimental evaluation are to experimentally assess the *effectiveness*, *efficiency* and *scalability* of the proposed modal specification verification method over workflow nets of growing size and complexity.

In the context of this experimentation, the proposed method is said to be effective over a given instance – a workflow net together with its modal specification – if and only if it is able to assign a verdict about the (in)validity of the given modal specification in an *admissible time.*

If the proposed method is effective over a given modal specification, its efficiency is its ability to return such a verdict in the least amount of time and memory. It follows that the proposed method is scalable if it is effective and efficient over workflow nets of growing size and complexity.

The secondary objective of this experimental evaluation is to compare the relative efficiency of the proposed method when employing two different constraint resolution approaches: Constraint Logic Programming and Satisfiability Modulo Theories.

5.2 Experimental Protocol

On the basis of the previously introduced objectives, the following experimental protocol has been designed.

About effectiveness assessment, an admissible time is arbitrary fixed to a reasonable value of 20 minutes, i.e. the time-out of constraint solver queries if fixed to 20 minutes. To evaluate the effectiveness of the proposed method, this protocol thus consists in gathering the proposed verification approach ability to assign a verdict to the instances of the considered data set in an admissible time of 20 minutes. If a verdict is assigned, the required time is also gathered in order to evaluate the proposed verification approach efficiency.

Furthermore, in order to compare the relative efficiency of the proposed method when employing two different constraint resolution approaches (CLP vs SMT), each instance of the considered data set has to be evaluated once using a CLP constraint solver, and once using a SMT solver.

Moreover, to make conclusion and feedback relevant and credible, and to be able to evaluate reliability as well as scalability of the methods, this information has to be calculated from a broad range of modal specifications and workflow nets.

Indeed, the type of modal specifications shall be taken into account because, to conclude about their validity, the verification method may require the computation of the over-approximation of the workflow nets executions or a full decomposition into segments. The size of the modal formula to be verified is also important since a larger formula may constrain further the system to be solved.

The proposed experimental protocol thus considers workflow nets of realistic and industrial size by evaluating workflow nets of size up to 1000 nodes. Moreover, not only the size of the workflow nets is considered but also their complexity by evaluating workflow nets of classes with a growing expressiveness (cf. Sect. 2.2). Therefore, to experimentally evaluate the effectiveness, efficiency and scalability of the proposed modal specification verification as well as the relative efficiency of the proposed method when employing two different constraint resolution approaches (CLP vs SMT) over instances of growing size and complexity, the following parameters are taken into account:

- Class of the workflow nets (*State machine, Marked graph, Free-choice, and Ordinary nets*)
- Size of the workflow nets ($25 * i$ *where* $i \in \{1, .., 40\}$)
- Type of modal specification (*Valid may-formula, Invalid may-formula, Valid must-formula, and Invalid must-formula*)
- Size of the modal formula (*5 and 15 literals*)

For each combination of the above parameters, a corresponding modal formula and a workflow net are randomly generated. This forms a data set of 1280 instances of growing size and complexity. Furthermore, in order to avoid statistical bias, five different data sets – i.e. a total of 6400 workflow nets and modal specifications – are randomly generated. The following section introduces the data set generation toolchain that has been used to carry the experimental protocol described in Sect. 5.2.

5.3 Data Set Generation Tools

To perform the proposed experimental protocol a benchmark generation toolchain has been implemented.

The purpose of these generation tools is to enable the generation of large benchmarks of realistic workflow nets of growing size and complexity together with their associated modal specifications according to the criteria defined in the experimental protocol of Sect. 5.2.

Figure 5.1 depicts the global architecture of the tooling developed to randomly generate modal formulae and workflow nets as defined by the protocol introduced in the previous section. The elements in the middle line of the figure represent the tools, while the elements above and below represent data files.

The entry point of the toolchain is an XML file that contains the specifications of the formula to be generated(1): its intended size as well as the probability at which the operator *and*, *or*, and *not* will be used by the Formula Generator(2) to produce a Formula(3) satisfying these given specifications. According to these specifications, the Formula Generator(2), starting from a single literal, recursively and randomly replaces an existing literal by (according to the given probability) either its negation, or its conjunction/disjunction with a newly introduced literal. Once the desired formula's size is reached, the produced formula is then saved in an XML file(3).

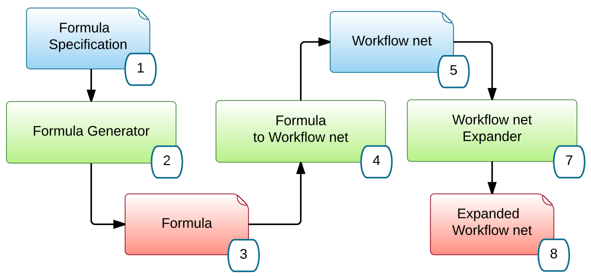


Figure 5.1. Architecture of the generation toolchain

The next step consists to produce an XML file describing a workflow net(5) satisfying (resp. not satisfying) the modal formula (resp. the negation of the modal formula) of the input formula interpreted as a must-formula. This computation is done using the Formula to Workflow net tool(4) that maps each operator of the input formula to its corresponding workflow net structure. Note that these structures can be different depending on the class of workflow nets that we intend to obtain. Further, we underline the fact that workflow nets generated this way are sound.

To illustrate this generation step, let us consider the *must*-formula $t_1 \wedge (t_2 \vee t_3)$. Figure 5.2 depicts the generated workflow net E_3 such that $E_3 \models_{must} t_1 \wedge (t_2 \vee t_3)$. In this figure, black boxes outline the nesting of the structures mapped from the operators of the input formula.

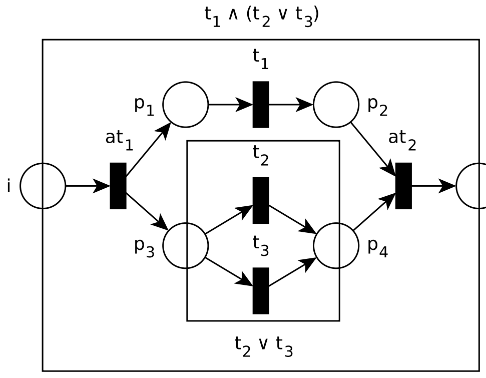


Figure 5.2. Workflow net E_3 generated for the modal specification $E_3 \models_{must} t_1 \wedge (t_2 \vee t_3)$

Finally, the produced workflow net(5) is expanded using the Workflow Expander tool(7). This tool expands the input workflow net(5) by randomly and successively applying synthesis rules (i.e. stepwise refinement) which add transitions and/or places while preserving the validity of the modal specification associated. The synthesis rules considered are the inverse rules of the reduction rules introduced in [5]. The result is an expanded workflow net saved as an XML file(8) satisfying or not the modal formula of the produced formula(3) interpreted as a may-formula or a must-formula. Note that, as the input workflow net is sound and the applied synthesis rules preserve soundness, the resulting expanded workflow net is also sound.

This expansion step is analogue to the benchmarks generation of Petri nets by random stepwise refinement proposed and discussed in [39]. The main difference between our expansion process and the generation process in [39] is the fact that the synthesis rules we consider generalize and therefore encompass and extend the synthesis rules that they considered thus producing more realistic and complex workflow nets. Indeed, experimental work presented in [5] has illustrated the effectiveness, i.e. the ability to reduce workflow nets, of the reduction rules it introduced over two benchmarks composed of a total of 1976 industrial workflow nets by providing a reduction factor of 82.2% on average. Further, it has demonstrated the ability of the proposed reduction rules to completely reduce all of the sound workflow nets of these benchmarks of industrial workflow nets thereby demonstrating the ability of the inverse of the proposed reduction rules, i.e. synthesis rules, to generate and produce such sound realistic and industrial workflow nets. This greatly underlines the relevance and quality of the workflow nets randomly generated by our toolchain.

Together the produced formula(3) and the produced expanded workflow net(8) form an instance of the data set as described by the experimental protocol in Sect. 5. It should be noticed that the XML output files of this generation toolchain can be directly used as input files of the verification toolchain introduced in Sect. 4.

6 Results and Feedback from Experiments

This section presents the experimental results obtained using the dedicated tool described in Sect. 4 when following the experimental protocol introduced in Sect. 5.2. All the executions have been computed on a computer featuring an Intel(R) Xeon(R) CPU X5650 @ 2.67GHz with 15.6 GiB of memory available. In total, these experimentations required about 1577 hours (65 days) of computation.

As said previously, various enumeration predicates could be used for SICStus, theoretically able to significantly reduce the resolution time on some instances. In order to determine the most adapted to our purpose, the following combinations have been used during the experiments for all instances of workflows:

- `leftmost`, `step` and `up` (the default)
- `leftmost`, `step` and `down`
- `leftmost`, `enum` and `up`
- `min`, `step` and `up`
- `ffc`, `step` and `up`
- `ffc`, `bisect` and `up`

Combined with `up` and either `step` or `bisect`, the `ffc` (standing for *First Fail Cut*) enumeration predicate tends to decrease the number of timeouts produced by SICStus compared to the other combinations tested. Since the `leftmost` predicate (the default one) essentially depends on the order in which the variables appear in the code, it mostly searches through the state space in a randomized manner. However, when using the `ffc` predicate, the variables with the smallest domain and with the most constraints on it are assigned first, which allows to search through the state space while staying guided by the structure of the network itself. Since most results were improved using this particular predicate, all SICStus experiments have been conducted using the `ffc`, `step` and `up` combination. Similarly, since all results were improved using the SMT tactic, this strategy has also been used for all Z3 experiments.

Table 6.1. State-Machine – Metrics

May-Valid/Must-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	492	513	656.5	778
	Z3	212.5	528	875	1219
Mean(ms)	SICStus	523	1097	2672	1126
	Z3	214	536	882	1240
#Timeout	SICStus	11(5.5%)	36(18%)	64(32%)	79(39.5%)
	Z3	0(0%)	0(0%)	0(0%)	116(58%)
Overall	SICStus	🟡	🟡	🟡	🟡
	Z3	🟢	🟢	🟢	🔴

Must-Valid/May-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	428	7805	364	379.5
	Z3	56	142	231	361
Mean(ms)	SICStus	6335	14550	385	420
	Z3	59	146	245	377
#Timeout	SICStus	141(70.5%)	196(98%)	197(98.5%)	192(96%)
	Z3	0(0%)	0(0%)	0(0%)	0(0%)
Overall	SICStus	🔴	🔴	🔴	🔴
	Z3	🟢	🟢	🟢	🟢

Table 6.2. Marked-Graph – Metrics

May-Valid/Must-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	462.5	496.5	655	868
	Z3	342.5	970	1691.5	357.5
Mean(ms)	SICStus	470	511	665	863
	Z3	354	1034	1937	356
#Timeout	SICStus	0(0%)	0(0%)	0(0%)	0(0%)
	Z3	0(0%)	0(0%)	76(38%)	192(96%)
Overall	SICStus	🟢	🟢	🟢	🟢
	Z3	🟢	🟢	🟡	🔴

Must-Valid/May-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	342.5	323.5	359	401.5
	Z3	84	185.5	285.5	381
Mean(ms)	SICStus	359	340	367	407
	Z3	86	186	292	407
#Timeout	SICStus	0(0%)	0(0%)	0(0%)	0(0%)
	Z3	0(0%)	0(0%)	0(0%)	0(0%)
Overall	SICStus	🟢	🟢	🟢	🟢
	Z3	🟢	🟢	🟢	🟢

🟢: Reasonable time, no time-out — 🟡: Reasonable time, # time-outs < 50% — 🔴: # time-outs > 50%

Table 6.3. Free-Choice – Metrics

May-Valid/Must-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	483	556	700	772
	Z3	251.5	634	1045	1053
Mean(ms)	SICStus	598	613	2734	6018
	Z3	266	634	1080	1069
#Timeout	SICStus	1(0.5%)	13(6.5%)	31(15.5%)	63(31.5%)
	Z3	0(0%)	0(0%)	0(0%)	188(94.4%)
Overall	SICStus	🟡	🟡	🟡	🟡
	Z3	🟢	🟢	🟢	🔴

Must-Valid/May-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	422	393	420.5	466
	Z3	64	136.5	248	350
Mean(ms)	SICStus	8559	2891	483	498
	Z3	66	149	254	374
#Timeout	SICStus	47(23.5%)	167(83.5%)	178(89%)	190(95%)
	Z3	0(0%)	0(0%)	0(0%)	0(0%)
Overall	SICStus	🟡	🔴	🔴	🔴
	Z3	🟢	🟢	🟢	🟢

Table 6.4. Ordinary – Metrics

May-Valid/Must-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	566.5	659.5	736	993
	Z3	484	679	1162	372
Mean(ms)	SICStus	3597	869	804	997
	Z3	1193	2045	1147	372
#Timeout	SICStus	64(32%)	126(63%)	133(66.5%)	139(69.5%)
	Z3	28(14%)	154(77.3%)	167(83.9%)	199(99.5%)
Overall	SICStus	🟡	🔴	🔴	🔴
	Z3	🟡	🔴	🔴	🔴

Must-Valid/May-Invalid

	↓Solver \ →Size	0-250	250-500	500-750	750-1000
Median(ms)	SICStus	474	325	391.5	476
	Z3	68	169	306	450
Mean(ms)	SICStus	10982	383	392	574
	Z3	73	176	326	506
#Timeout	SICStus	75(37.5%)	194(97%)	196(98%)	195(97.5%)
	Z3	0(0%)	0(0%)	0(0%)	0(0%)
Overall	SICStus	🟡	🔴	🔴	🔴
	Z3	🟢	🟢	🟢	🟢

🟢: Reasonable time, no time-out — 🟡: Reasonable time, # time-outs $< 50\%$ — 🔴: # time-outs $> 50\%$

To provide relevant feedback regarding the initial challenges given in Sect. 1, the obtained results are discussed by distinguishing two different categories of modal specifications: may-valid and must-invalid specifications, and may-invalid and must-valid specifications.

Note that the algorithm given in Sect. 3.2 is applied to all generated workflows and specifications, no matter what type of specification is being verified. Thus, our implementation first checks if an over-approximation is sufficient to conclude about the validity or the invalidity of a specification and, only if it is not the case, computes an under-approximation before concluding. Indeed, using the verification algorithm given in Sect. 3.2, most may-valid and must-invalid modal specifications can be verified by using only an over-approximation of correct executions of the workflow. This over-approximation is less complex than the under-approximation that must very often be computed to verify may-invalid and must-valid modal specifications.

In this context, even though may-valid and must-invalid specifications express two different behaviours (i.e. a may-valid specification is not necessarily a must-invalid specification), most of the specifications of this category may only require the computation of the over-approximations of correct executions of the workflow. On the contrary, even though may-invalid and must-valid specifications express two opposite behaviours (i.e. a may-invalid specification is never a must-valid specification and vice versa), most of these specifications often require the costly computation of under-approximations of correct executions.

We also categorise the results according to the different classes of workflow nets considered in our experimental protocol. The average execution times reported in the following subsections have been computed without considering time-outs. Thus, since time-outs may have occurred, similar average execution times do not always induce similar performances from both solvers. Nonetheless time-outs are stated and discussed separately. Finally, for clarity, all time-outs have been withdrawn from the plots but systematically taken into account in our feedback. The interested reader can also study the complete data sets and results given at `https://figshare.com/articles/Benchmarks/4708135`.

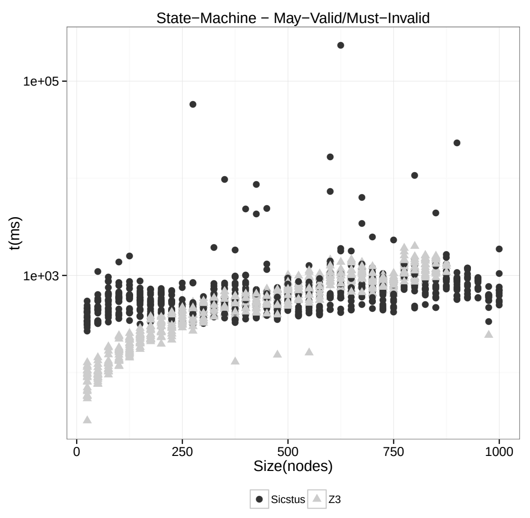
Tables 6.1, 6.2, 6.3 and 6.4 summarize the median verification times, average verification times, number of time-outs and the overall appreciation of the results obtained over the different studied workflow net classes.

Figures 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7 and 6.8 depict the plots displaying the verification times spent by SICStus and Z3 as well as the distribution of time-outs with respect to the considered workflow nets sizes for each class of workflow nets and types of modal specifications.

The obtained results are now reported and the most important feedback for each class of workflow nets.

6.1 State-Machine Workflow Nets Verification

May-Valid and Must-Invalid specifications.



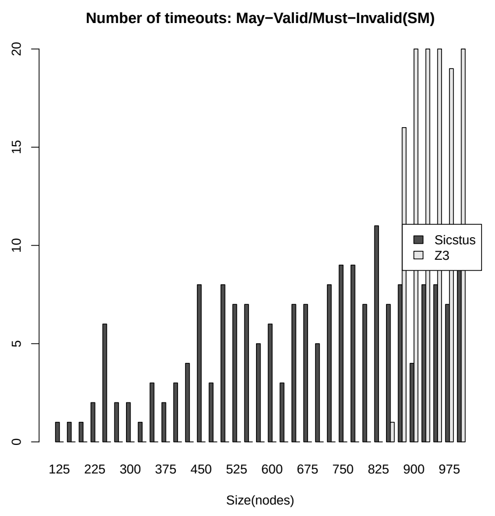
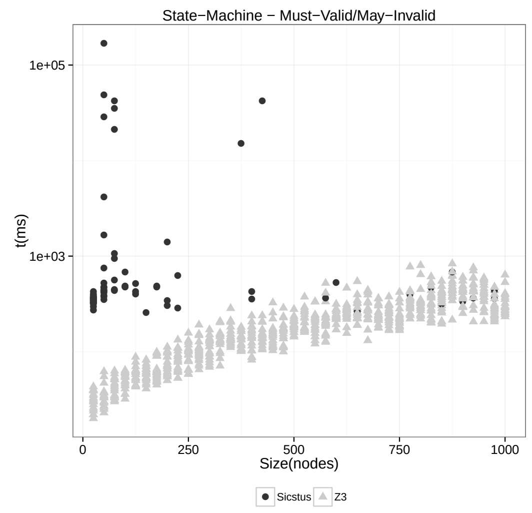


Figure 6.1. State-Machine – May-Valid/Must-Invalid

Observation. Whenever they were able to conclude within the imposed time limit, both solvers were able to conclude in a comparable and reasonable time. Indeed, Z3 median execution time was 582ms whereas SICStus median execution time was 586ms. However we observe that SICStus is progressively overwhelmed as shown by the growing number of time-outs with respect to the size of considered workflow nets. Furthermore, regarding Z3 performance, we observe a clear workflow nets size bound above which it cannot conclude within the imposed time limit. Indeed, for workflow nets of size greater than 850 nodes Z3 is not able to provide results whereas SICStus is still able to perform well.

Must-valid and May-Invalid specifications.



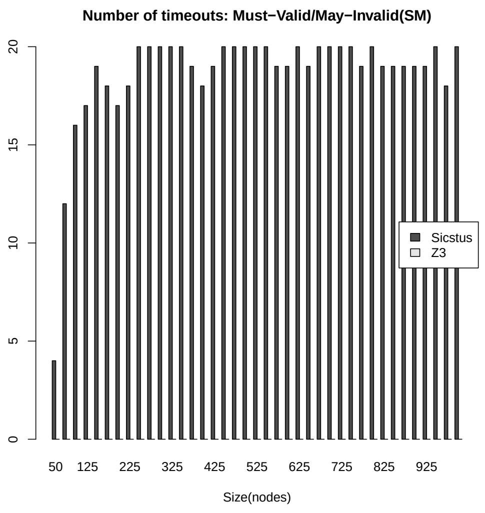


Figure 6.2. State-Machine – Must-Valid/May-Invalid

Observation. On the one hand, we observe, with the exception of a few instances, that SICStus is not able to provide answers within the imposed time limit. On the other hand, we observe that Z3 is consistently able to complete its computations in a very reasonable time (i.e. 207ms on average and in less than a second in the worse case) on instances of size up to 1000 nodes. These results clearly show that Z3 performs better than SICStus for this type of modal specifications.

Synthesis. Over the class of State-Machines we conclude that SICStus is clearly overwhelmed due to the high number of choice points arising from the structure of State-Machine workflow nets of size greater than 100 nodes.

Further, we note that, when considering State-Machine workflow nets the number of states of the underlying search space is equal to the number of states of the reachability graph. The modelling of an execution of a state machine workflow net requires the resolution of the constraint systems associated with as many segments as the number of transitions in the execution modelled (i.e. there is a single transition per segment). More precisely, it therefore often requires the resolution of a large amount of segments which is very costly in terms of time and memory. In such (limit) cases, it is therefore expected that explicit generation and exploration of the reachability/coverability graphs for basic model checking can be more efficient than the proposed method based on constraint solving.

Furthermore, we conclude that Z3 is able to efficiently verify may-valid and must-invalid modal specification of workflow net of size up to 850 nodes and is suddenly overwhelmed above this size. However, when verifying must-valid and may-invalid modal specifications, Z3 is able to efficiently conclude within less than a second over workflow nets of size up to 1000 nodes.

We conclude from these results that the SMT approach seems to be efficient and more suited for the modal specifications verification over State-Machine workflow nets.

6.2 Marked-Graph Workflow Nets Verification

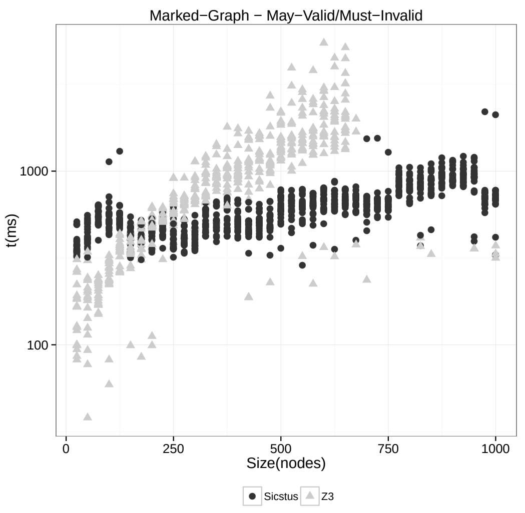
May-Valid and Must-Invalid specifications.

Observation. We observe that SICStus is consistently and efficiently able to conclude within the imposed time limit, more specifically in less than 5 seconds (627ms on average). Regarding Z3 performances, we observe that the SMT solver is more efficient than SICStus for workflows of size lower than 250 nodes and is less efficient than SICStus for workflow nets of size ranging from 250 to 650 nodes. Above this limit however, we also observe that Z3 cannot (except for a few instances) determine the validity of modal specification over workflow nets of size greater than 650 nodes.

Must-valid and May-Invalid specifications.

Observation. Over all instances of size up to 1000 nodes, both solvers are able to efficiently and consistently conclude within about 1 second (SICStus in 368ms and Z3 in 243ms on average). Overall, we also observe that Z3 is slightly more efficient than SICStus notably over instances of size lower than 750 nodes.

Synthesis. Over the class of Marked-Graphs, we can conclude that SICStus and Z3 perform similarly despite a slight advantage (in terms of performance) for Z3 when verifying modal specification of workflow nets of size up to 650 nodes. However, for workflow nets of size greater than 650, SICStus seems to perform better when verifying May-Valid and Must-Invalid specifications, while



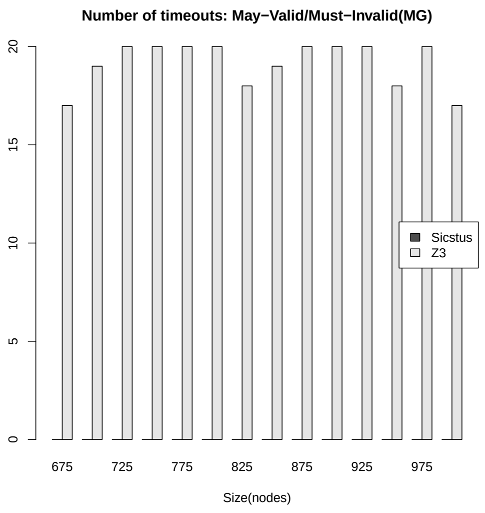


Figure 6.3. Marked-Graph – May-Valid/Must-Invalid

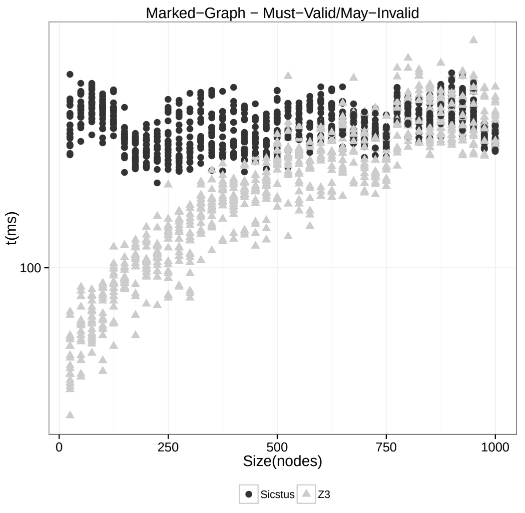


Figure 6.4. Marked-Graph – Must-Valid/May-Invalid

Z3 is not able to conclude. A further investigation has shown that, over Marked-Graphs, Z3 is more effective than SICStus with regard to the computation of the over-approximation used by the verification method, while SICStus is more effective than Z3 with regard to the computation of the segment (only one) needed to conclude whenever the over-approximation is not sufficient.

6.3 Free-Choice Workflow Nets Verification

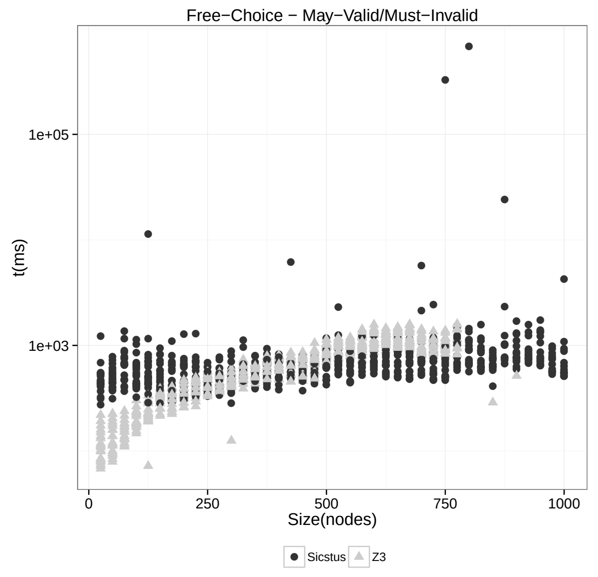
May-Valid and Must-Invalid specifications.

Observation. The observations made for this class of workflow nets are similar to the ones made for the class of State-Machines. Overall, over workflow nets of size up to 800 nodes, both solvers are able to conclude in a consistent and efficient manner (with a median execution time of 630ms for SICStus and 641ms for Z3) despite a very small performance advantage for Z3 over instances of size lower than 250 nodes. However, regarding Z3, we observe a similar bound in terms of size (around 750 nodes) over which Z3 is overwhelmed while SICStus is still able to obtain results.

Must-valid and May-Invalid specifications.

Observation. Once again, the observation about the results obtained over Free-Choice workflow nets are similar to the ones obtained over State-Machine workflow nets. Likewise, on the one hand, we observe, with the exception of a few instances (albeit more than in the case of State-Machine workflow nets), that SICStus is not able to provide answers within the imposed time limit. On the other hand, we observe that Z3 is consistently able to complete its computations in a very reasonable time (i.e. 201ms on average and in less than a second at worse) on instances of size up to 1000 nodes. These results clearly show that Z3 outperforms SICStus for this type of modal specifications. After investigation, these results stem from the fact that the verification of such modal specifications mostly relies on the results of an over-approximation for which Z3 performs apparently far better off.

Synthesis. Over the class of Free-Choice workflow nets, we observe that Z3 performs better than SICStus for workflow nets of size up to 750 nodes. Over this limit, when verifying May-Valid and Must-Invalid modal specifications, Z3 is overwhelmed, while SICStus is still able to conclude on some instances. This limit can be explained by the fact that Free-Choice workflow nets combine the expressiveness of State-Machines and Marked Graphs, therefore this limit is consistent with the limit observed over the class of State Machine workflow nets. Overall, we thus conclude from these obtained results that the SMT approach seems to be more suited for the modal specifications verification over Free-Choice workflow nets.



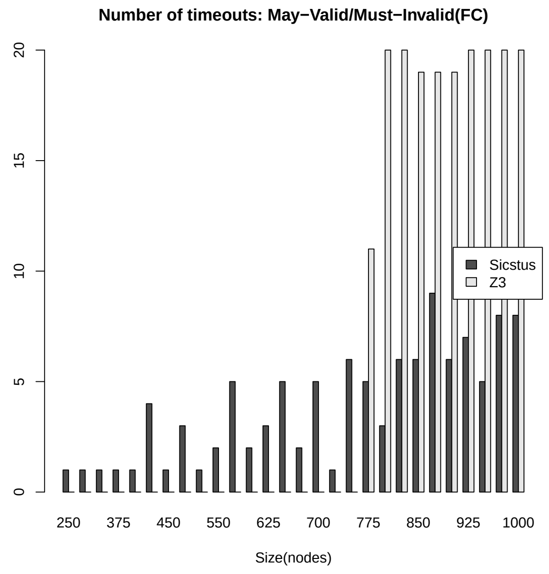
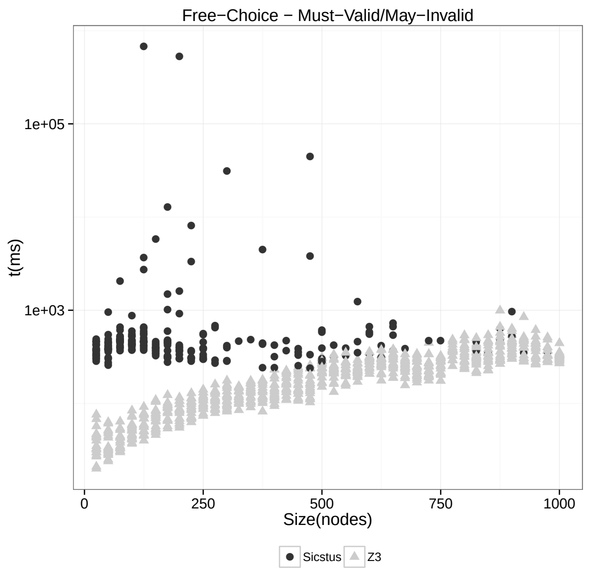


Figure 6.5. Free-Choice – May-Valid/Must-Invalid



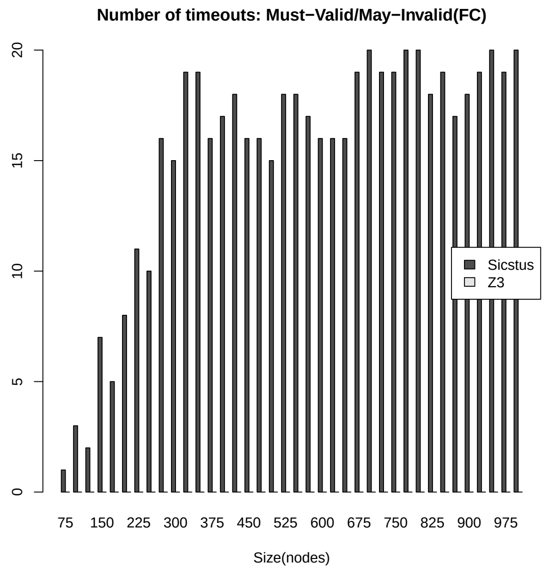


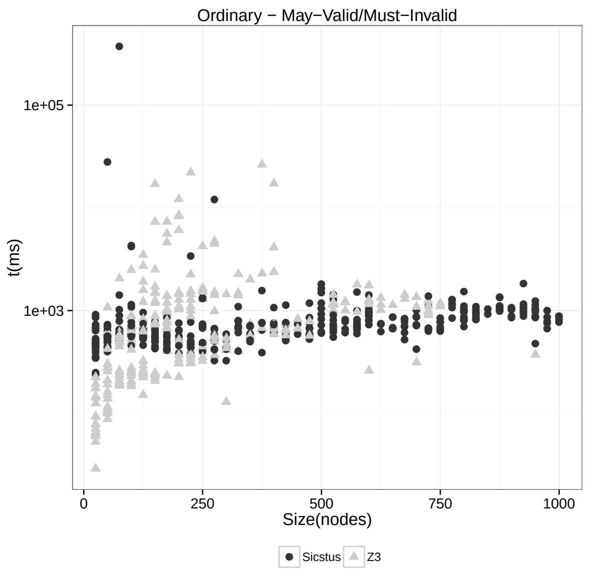
Figure 6.6. Free-Choice – Must-valid/May-Invalid

May-Valid and Must-Invalid specifications.

Observation. Over ordinary workflow nets, both solvers, whenever they were able to conclude within the imposed time limit, seem to behave similarly although we note a very slight advantage for Z3 (with a median execution time of 684ms for SICStus and 626ms for Z3). Indeed, they both seem to be gradually unable to conclude as the size of the considered workflow nets grows. Furthermore, once again, as it is the case over Free-Choice and State-Machine workflows nets, Z3 is not able to conclude (except on a single instance) about workflow nets of size greater than 750 nodes.

Must-valid and May-Invalid specifications.

Observation. The results obtained for the verification of must-valid and may-invalid modal specifications over Ordinary workflow nets are very similar to those obtained for the verification of must-valid and may-invalid modal specifications over Free-Choice workflow nets. Indeed, we observe that SICStus, for most instances, is unable to conclude within the imposed time limit, whereas Z3 always produces a result within about two second (271ms on average). As for the previous classes, Z3 indeed performs better than SICStus to compute the over-approximation constraints, which are often sufficient to conclude.



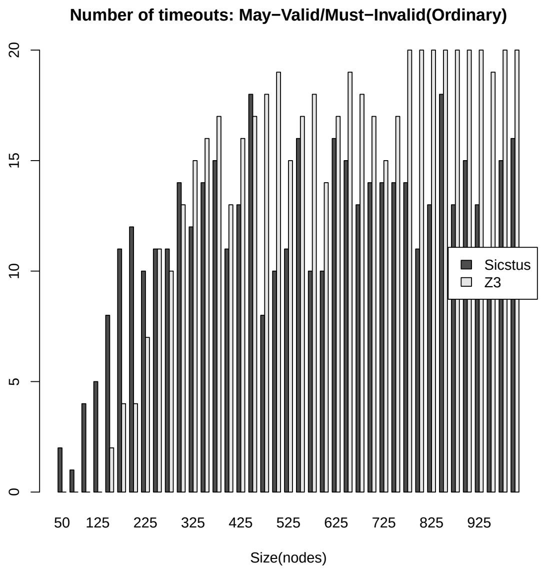
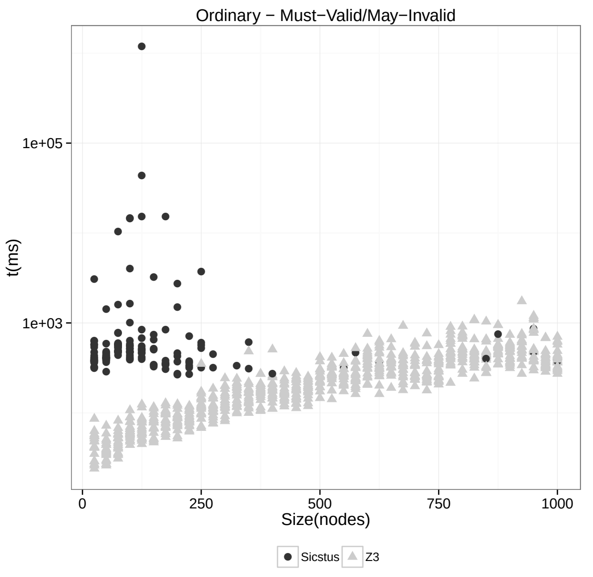


Figure 6.7. Ordinary Workflow – May-Valid/Must-Invalid



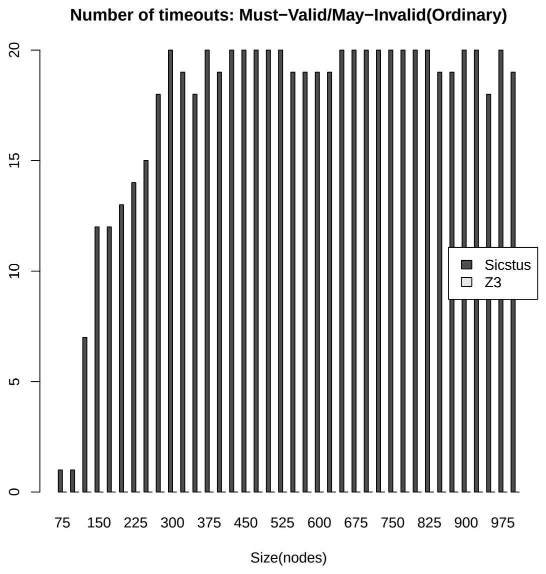


Figure 6.8. Ordinary Workflow – Must-valid/May-Invalid

Synthesis. Over the class of Ordinary workflow nets, we observe that Z3 performs better than SICStus with workflows of size up to 750 nodes. Over this limit, both solvers are for most instances unable to conclude when verifying may-valid and must-invalid modal specifications. Z3, however, is always able to produce a result when verifying must-valid and may-invalid modal specifications, whereas SICStus is almost always unable to produce an answer. Overall, we thus conclude that the SMT approach seems to be more suited for the verification of modal specifications over Ordinary workflow nets.

The next section summarizes the lessons learned and the benefits noticed from these experiments according to the initial challenges.

6.5 Lessons Learned from Experience

Effectiveness. Overall the modal specification verification method based on constraints solving and its implementation appear to be rather effective over workflow nets of size up to a 1000 nodes. More precisely, on the one hand, our experiments highlight the very good effectiveness of the verification method when considering the verification of must-valid/may-invalid (resp. may-valid/must-invalid) modal specifications over workflow nets of size up to a 1000 nodes (resp. up to a 750 nodes). On the other hand, when considering may-valid/must-invalid modal specifications of workflow nets of size greater than 750 nodes, both solvers tend to have difficulties to produce a verdict.

Efficiency. The presented toolchain, based on the verification method proposed in [3] and the underlying constraint solvers–Z3 and SICStus, has been shown to be very efficient for the intended verification computation. Indeed, whenever the toolchain is able to conclude about the (in)validity of modal specification, it is usually able to do so in less than a few seconds independently of the considered class of workflow nets and modal specification type. Such a short analysis time means that this procedure could be automatically applied by integrated development environment to provide feedback as well as useful diagnostic information during workflow nets development. Regarding the performances with respect to formula size, we do not observe any consistent nor significant variations.

Scalability. On the basis of the results, we can confidently state that the verification method proposed in [3] is scalable in terms of modal specification size and workflow net complexity, as well as regarding their size (up to 750 nodes at least). Such results enable the authors to confidently state that the verification of modal specifications is feasible within real-life industrial settings and can therefore benefit workflow engineers in their workflow validation and verification activities.

Further, we observe that, when using the proposed verification method, the size of the reachability graph of the studied workflow net is not directly influencing the results. Instead, we observe that the size of the underlying search space whose state are markings and whose transitions are not single basic transitions but maximal segments is directly influencing the results. Notably, a large reachability graph does not necessary imply a large underlying search space, e.g. when considering Marked graph workflow nets.

SMT vs CLP. According to these experiments, we can infer that the SMT approach (computed using Z3) generally performs significantly better than the CLP one (computed using SICStus). However, they also highlight that the CLP approach performs better, especially when verifying modal specifications over Marked-Graph workflow nets. Indeed, we observed that the CLP approach is less efficient than the SMT approach when the number of choice points increases as shown by the results over State-Machine workflow nets. It stems from the labeling done after constraints propagation by CLP solvers: an exponential number of backtracking steps may occur w.r.t. the number of pending choice points. Nonetheless, choosing more specific heuristics can often greatly improve performance. Indeed, our experiments have highlighted that among a large number of heuristics seen in Sect. 5.2, the most efficient, and therefore the most effective, seems to be first fail cut (`ffc`), which in our case prioritizes the valuation of the most constrained variables instead of using an unplanned order (i.e. default order).

Table 1. Bench1 – Must-Valid

	t(s)		
#Nodes	SM	FC	Ordinary
25	0.6	0.1	1.1
50	0.8	0.2	0.4
75	1.0	106.3	201.4
100	28.8	1070.7	*
125	70.6	*	*
150	322.2	*	*
175-1000	*	*	*

On the basis of these experimental results we are able to draw the following modal specification verification strategy. When verifying may-valid/must-invalid modal specifications, whenever the size of the workflow nets under analysis is not greater than 750 nodes, we advise to use Z3 with a low time-out (10 seconds). If this is not conclusive, we then recommend using SICStus. Otherwise, when considering workflow nets of size greater than 750 nodes we advise the use of SICStus.

Furthermore, when verifying must-valid/may-invalid modal specifications we recommend using Z3.

Overall, we observe that, for instances of size up to a 1000 nodes, a result is generally produced within a few seconds (10 seconds) or requires a unreasonable time (greater than 20 minutes in the presented experiments and more than 24 hours according to additional investigations). For that matter, if a result is not obtained within this short duration then switching to an alternative solver is preferable.

Comparison with model checking approach. As stated in Section 2 modal specifications are a proper subset of CTL, it is therefore possible to check any modal specifications of a given workflow net using any CTL Petri net model checker (see Appendix A).

As a point of reference, we provide new additional experimental results obtained using traditional model checker to further support the relevance as well as the advantages in terms of efficiency and scalability of the constraint solving based verification approach studied in this paper when verifying modal specifications.

Table 1 reports on the execution time necessary to verify the validity of Must-Valid modal specification over State-Machine, Free-Choice, and Ordinary workflow nets of growing size issued from one of the previously used benchmarks (Bench1). These results have been obtained using LoLa [30] – a state of the art CTL Petri net model checker, winner of the 2017 Model Checking Contest (CTL Formulas Category) [20]. In Table 1 cells marked by an asterisk (*) represent computation that ran out of memory before completion.

We observe that LoLa appears efficient when verifying modal specifications over workflow nets of small size (< 100 nodes) but gets rapidly overwhelmed when verifying modal specifications over workflow nets of greater size. It is notably unable to conclude about the validity of modal specifications over workflow nets of size greater than 175 nodes, mainly due to memory constraints. These results clearly demonstrate the benefit of the constraint solving approach evaluated in the paper, both in terms of efficiency and scalability. This can be explained by the fact that LoLa is a general purpose model checker capable of verifying a greater range of properties, whereas the method proposed in this paper is tailored to the verification of modal specifications.

7 Related Work

Verifying properties over business processes has been widely investigated using approaches based on Petri nets. Among them, workflow nets constitute a suited class for modelling business process as reported, among others, in [35]. Thus, approaches and tools [3, 2, 40] have emerged to verify properties over these workflow nets and, as a consequence, over the business processes they model. However, regarding verification of such WF-nets, the reachability problem, arising with most problems related to workflow nets, and proved to be an EXPSPACE problem in [18], is the key problem that all approaches are facing. To tackle this issue, the formal method given in [3] and experimented in a further and wider manner in this paper, is inspired from [41, 40].

Regarding verification methods, some research results have also been proposed to express and verify properties against a given system. Let us quote [23] where the expression of properties with modalities is investigated for automata/transition systems, and also [21] where they are studied for Petri nets. In this context, the great expressiveness of modalities makes them popular and relevant for precisely describing a possible or necessary behavior over a system. This paper precisely evaluates two CSP resolution techniques to verify modal specifications over large-scale WF-nets, using the approach given in [3].

Formal verification methods based on constraints solving have been studied intensively, with most concrete implementations using the SMT or CLP approaches. On the one hand, for example, SMT has been used in [25] for checking the reachability of bounded Petri nets, as well as in [28] for verifying properties of business processes where execution paths are modelled as constraints. On the other hand, CLP has been also extensively experimented to verify business processes [19] as well as Petri nets [40]. In a very similar way, a CLP approach has been used in [31] to detect the presence of structures potentially leading to deadlocks in Petri nets. The present paper has the originality to compare SMT and CLP resolution methods to implement the constraint-based verification approach in [3].

8 Conclusion

This paper aimed to evaluate the efficiency and the scalability of the constraint-based approach, proposed in [3] and [4], able to verify modal specifications over workflow nets. To reach this objective, a dedicated toolchain, integrating both SMT and CLP solving, has been developed to support the full verification process. This toolchain made it possible to conduct an accurate experimentation to validate and evaluate the approach and to provide a precise comparison regarding the scalability of the both above-mentioned resolution methods, i.e. SMT and CLP, to verify modal specifications over workflow nets using constraint solving.

Compared to previous work [6, 7], the experiments have been carried out with a finer and broader range of four classes of workflow nets that cover a wide spectrum of model variations. The present paper has indeed reported on a new extensive experimentation campaign involving more workflow nets (6400 against 960 previously), classified according to their size (size up to 1000 nodes against 500 previously), and conducted using a time-out of 20 minutes per resolution (against 10 minutes previously).

The obtained experimental results empirically demonstrate that the verification method is efficient and scalable over workflow nets of size up to 1000 nodes. In general, the SMT approach performs significantly better than the CLP approach, except when verifying modal specifications over conflict-free workflow nets, i.e. Marked-Graphs. However, the results also highlight the efficiency of the CLP approach when verifying modal specifications over Marked-Graphs. Moreover, on the basis of the experimental results, we have drawn a modal specification verification strategy that combines SMT and CLP methods according to the features of the workflow net under verification, and the nature of the modal specification to be verified.

As a future work, we would like to investigate in a further way and evaluate such strategies, mixing both SMT and CLP methods, in order to embrace the benefits from each of them and to take advantage of the potential synergy. Moreover, we plan to apply our approach to more real-life industrial workflows to confirm its efficiency, and to gather more feedback to propose possible further improvements.

References

1. F. Bellegarde, C. Darlot, J. Julliand, and O. Kouchnarenko. Reformulation: a way to combine dynamic properties and b refinement. In *FME*, volume 2021, pages 2–19. Springer, 2001.
2. H.H. Bi and J.L. Zhao. Applying propositional logic to workflow verification. *Information Technology and Management*, 5(3-4):293–318, 2004.

3. H. Bride, O. Kouchnarenko, and F. Peureux. Verifying modal workflow specifications using constraint solving. In *Proc. of Int. Conf. on Integrated Formal Methods (IFM'14)*, volume 8739 of *LNCS*, pages 171–186, Bertinoro, Italy, September 2014. Springer.
4. H. Bride, O. Kouchnarenko, and F. Peureux. Constraint solving for verifying modal specifications of workflow nets with data. In *Proc. of 10^{th} Int. Ershov Informatics Conf. - Perspectives of Syst. Informatics (PSI'15)*, volume 9609 of *LNCS*, pages 75–90, Kazan, Russia, August 2015. Springer.
5. H. Bride, O. Kouchnarenko, and F. Peureux. Reduction of workflow nets for generalised soundness verification. In *Proc. of the 18^{th} Int. Conf. on Verification, Model-Checking, and Abstract Interpretation (VMCAI'17)*, volume 10145 of *LNCS*, pages 91–111, Paris, France, January 2017. Springer.
6. H. Bride, O. Kouchnarenko, F. Peureux, and G. Voiron. Comparaison des approches SMT et CSP appliquées à la vérification de réseaux workflows. In *Actes des 15èmes journées sur les Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'16)*, pages 11–12, Besançon, France, June 2016.
7. H. Bride, O. Kouchnarenko, F. Peureux, and G. Voiron. Workflow nets verification: SMT or CLP? In *Proc. of the 21^{st} Int. Wsh. on Formal Methods for Industrial Critical Syst. and Automated Verification of Critical Syst. (FMICS-AVoCS'16)*, volume 9933 of *LNCS*, pages 1–17, Pisa, Italy, September 2016. Springer.
8. M. Carlsson et al. *SICStus Prolog user's manual (Release 4.2.3)*. Swedish Institute of Computer Science, Kista, Sweden, October 2012.
9. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Programming Languages and Syst. (TOPLAS)*, 8(2):244–263, 1986.
10. L. De Moura and N. Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
11. L. De Moura and N. Bjørner. Satisfiability modulo theories: Introduction and applications. *Commun. ACM*, 54(9):69–77, September 2011.
12. P. K. T. Edward. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press, 1993.
13. D. Elhog-Benzina, S. Haddad, and R. Hennicker. Refinement and asynchronous composition of modal petri nets. *Trans. Petri Nets and Other Models of Concurrency*, 5:96–120, 2012.
14. M. V. Espada and J. van de Pol. Accelerated modal abstractions of labelled transition systems. In *Int. Conf. on Algebraic Methodology and Software Technology*, pages 338–352. Springer, 2006.
15. D. Fahland, C. Favre, J. Koehler, N. Lohmann, H. Völzer, and K. Wolf. Analysis on demand: Instantaneous soundness checking of industrial business process models. *Data & Knowledge Engineering*, 70(5):448–466, 2011.
16. S. Goedertier and J. Vanthienen. Designing compliant business processes with obligations and permissions. In *Business process management workshops*, pages 5–14. Springer, 2006.
17. G. Governatori, Z. Milosevic, and S. Sadiq. Compliance checking between business processes and business contracts. In *Enterprise Distributed Object Computing Conf., 2006. EDOC'06. 10th IEEE Int.*, pages 221–232. IEEE, 2006.
18. S. Haddad. Decidability and complexity of Petri net problems. *Petri Nets: Fundamental Models, Verification and Applications*, pages 87–122, 2009.
19. M. Kleine and T. Göthel. Specification, verification and implementation of business processes using CSP. In *TASE*, pages 145–154. IEEE Computer Society, 2010.
20. F. Kordon, H. Garavel, L. M. Hillah, F. Hulin-Hubard, B. Berthomieu, G. Ciardo, M. Colange, S. Dal Zilio, E. Amparore, M. Beccuti, T. Liebke, J. Meijer, A. Miner, C. Rohr, J. Srba, Y. Thierry-Mieg, J. van de Pol, and K. Wolf. Complete Results for the 2017 Edition of the Model Checking Contest. http://mcc.lip6.fr/2017/results.php, June 2017.
21. O. Kouchnarenko, N. Sidorova, and N. Trcka. Petri Nets with May/Must Semantics. In *Wsh. on Concurrency, Specification, and Programming - CS&P 2009*, volume 1, Kraków-Przegorzaly, Poland, September 2009.
22. K. G. Larsen. Modal specifications. In *Proc. of the Int. Wsh. on Automatic Verification Methods for Finite State Syst.*, pages 232–246, London, UK, 1990. Springer-Verlag.
23. K.G. Larsen and B. Thomsen. A modal process logic. In *Logic in Computer Science, 1988. LICS '88., Proc. of the Third Annual Symposium on*, pages 203–210, 1988.
24. Ernst W Mayr. An algorithm for the general petri net reachability problem. *SIAM J. on computing*, 13(3):441–460, 1984.
25. G. Monakova, O. Kopp, F. Leymann, S. Moser, and K. Schäfers. Verifying business rules using an SMT solver for BPEL processes. In *BPSC*, volume 147 of *LNI*, pages 81–94. GI, 2009.
26. T. Murata. Petri nets: Properties, analysis and applications. *IEEE*, 77(4):541–580, April 1989.
27. C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962.
28. A. Pólrola, P. Cybula, and A. Meski. Smt-based reachability checking for bounded time Petri nets. *Fundam. Inform.*, 135(4):467–482, 2014.
29. K. Salimifard and M. Wright. Petri net-based modelling of workflow systems: An overview. *European J. of operational research*, 134(3):664–676, 2001.
30. Karsten Schmidt. Lola a low level analyser. In *International Conference on Application and Theory of Petri Nets*, pages 465–474. Springer, 2000.
31. S. Soliman. Finding minimal p/t-invariants as a csp. In *Proc. of the 4th Wsh. on Constraint Based Methods for Bioinformatics WCB*, volume 8, 2008.
32. I. Suzuki and T. Murata. A method for stepwise refinement and abstraction of petri nets. *J. of computer and syst. sciences*, 27(1):51–76, 1983.
33. W. M. P. Van Der Aalst. Three good reasons for using a petri-net-based workflow management system. In *Proc. of the Int. Working Conf. on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201. Cambridge, Massachusetts, 1996.
34. W. M. P. van der Aalst. Verification of workflow nets. In *Proc. of the 18th Int. Conf. on Application and Theory of Petri Nets*, ICATPN '97, pages 407–426, London, UK, 1997. Springer-Verlag.

35. W. M. P. van der Aalst. The application of Petri nets to workflow management. *J. of Circuits, Syst. and Computers*, 08(01):21–66, 1998.
36. W. M. P. Van Der Aalst. Woflan: a petri-net-based workflow analyzer. *Syst. Anal. Model. Simul.*, 35(3):345–358, 1999.
37. W. M. P. van der Aalst, K. M. van Hee, A. H. M. ter Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn. Soundness of workflow nets: classification, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011.
38. K. Van Hee, N. Sidorova, and M. Voorhoeve. Soundness and separability of workflow nets in the stepwise refinement approach. In *ICATPN*, volume 2679, pages 337–356. Springer, 2003.
39. K. M. Van Hee and Z. Liu. Generating benchmarks by random stepwise refinement of Petri nets. In *ACSD/Petri Nets Workshops*, pages 403–417, 2010.
40. H. Wimmel and K. Wolf. Applying CEGAR to the Petri net state equation. *Logical Methods in Computer Science*, 8(3), 2012.
41. P. Y. H. Wong and J. Gibbons. A process-algebraic approach to workflow specification and refinement. In *Proc. of the 6th Int. Conf. on Software Composition*, SC'07, pages 51–65, Berlin, Heidelberg, 2007. Springer-Verlag.

A Appendix - Verifying Modal Specification Using Model Checking

In this appendix we describe the methodology used to verify modal specifications using a CTL Petri net model checker such as LoLa [30].

As stated in Section 2 modal specifications are a proper subset of CTL, it is therefore possible to check any modal specifications of a given workflow net using any CTL Petri net model checker. To this end, the considered workflow net needs to be transformed into an equivalent Petri net such that the validity of any modal specifications is equivalent to the validity of the the corresponding CTL formulae. The aim of this transformation is to introduce for each transition t a new place p_t which is marked if and only if t has been fired at least once during an execution marking the final place o.

To produce a Petri net $\tilde{N}$ from a workflow net $N = \langle P, T, F \rangle$ the transformation proceeds as follows. For each transition $t \in T$ the transformation introduces two new places respectively denoted f_t and p_t. Further, the transformation then replaces each transition t by two transitions respectively denoted t_f and t_e such that:

- ${}^\bullet t_f = {}^\bullet t \cup \{f_t\}$
- $t_f^\bullet = t^\bullet \cup \{p_t\}$
- ${}^\bullet t_e = {}^\bullet t \cup \{p_t\}$
- $t_e^\bullet = t^\bullet \cup \{p_t\}$

Note that if the state space of N is finite, then the state space of $\tilde{N}$ is finite too.

The initial marking of $\tilde{N}$ is the marking assigning a single token to place i and places f_t where $t \in T$ (and none to other places).

By construction, for any execution σ of N, there exist a corresponding execution $\tilde{\sigma}$ of $\tilde{N}$ obtained by replacing, for every transition $t \in T$, the first occurrence of t by t_f and the following occurrences of t by t_e. Conversely, for any execution $\tilde{\sigma}$ of $\tilde{N}$ there exists a corresponding execution σ of N obtained by replacing, for every transition $t \in T$, all occurrences of t_f and t_e by t.

Further, given a modal formula $f \in S$ of the workflow net N, we define $CTL(f)$ as the formula obtained after replacing, for every transition $t \in T$, the corresponding terminal symbols of the modal formula f by $p_t = 1$.

Consequently, as for each transition t the new place p_t is marked if and only if t has been fired at least once during an execution, we have:

- $N \models_{may} f \Leftrightarrow \tilde{N} \models EF(o = 1) \Rightarrow (CTL(f))$
- $N \models_{must} f \Leftrightarrow \tilde{N} \models AF(o = 1) \Rightarrow (CTL(f))$

This enables the verification of modal specifications described in this paper using CTL model checker, and consequently this link makes it possible to compare verification approaches by constraint solving and by model checking.