Workflow Nets Verification: SMT or CLP?

Hadrien Bride, Olga Kouchnarenko, Fabien Peureux, and Guillaume Voiron

Institut FEMTO-ST – UMR CNRS 6174, University of Franche-Comté
16, route de Gray, 25030 Besançon, France
{hbride,okouchna,fpeureux,gvoiron}@femto-st.fr

Abstract. The design and the analysis of business processes commonly relies on workflow nets, a suited class of Petri nets. This paper evaluates and compares 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 presents the experimental protocol designed to evaluate and compare the scalability and efficiency of both resolution approaches. Thirdly, the paper reports on the obtained results and discusses the lessons learned from these experiments.

Keywords: 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. Workflows constitute a convenient way for analysts to describe the business processes in a formal and graphical manner. Intuitively, a workflow system describes the set of possible runs of a particular system/process. Furthermore, workflow analysts are required to express and to verify specific properties over the workflows they designed to 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. As in [1], the validity of a modal specification can be inferred from the satisfiability of a corresponding constraint system, by using Constraint Logic Programming (CLP). Besides the theoretical assessment of the approach, a proof-of-concept toolchain has enabled to successfully evaluate its effectiveness and reliability. However, as advocated in [1], these first encouraging experimental results need to be confirmed by extensive experimentation, in particular to definitively assess the scalability and the efficiency of the approach. This paper precisely investigates these issues: it aims to empirically (1) assess the scalability, (2) evaluate the efficiency of the verification approach by (3) comparing two resolution methods: Satisfiability Modulo Theory (SMT) and CLP over Finite Domains to solve the Constraint Satisfaction Problem (CSP) that represents the modal specifications to be verified.

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 [2] 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). Basically, SMT solvers aim to generate counter-examples [3] 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.

Layout of the paper and contributions. Section 2 briefly recalls common concepts and standard notations concerning workflow nets as well as the key aspects of the formal method given in [1] for verifying modal specifications over workflow nets. Afterwards, Sect. 3 defines an 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. To achieve this goal, a mature toolchain has been developed to automatically produce, from a workflow net and its modal specification, a constraint system whose satisfiability can then be checked using either CLP or SMT. Section 4 reports on the experimental results obtained using the experimental protocol. The lessons learned as well as the reported feedback constitute the main contribution of this paper. Finally, Sect. 5 suggests directions for future work.

2 Preliminaries

This section presents workflow nets [4], modal specifications as well as the verification method proposed in [1].

2.1 Workflow Nets

Workflow nets (WF-nets) [4] are a special case of Petri nets. They allow the modelling of complex workflows exhibiting concurrencies, conflicts, as well as causal dependencies of activities. The different activities are modelled by transitions, while causal dependencies are modelled by places and arcs. For instance, the Petri net depicted in Fig. 1 is a workflow net.

Definition 1 (Workflow net [4]). *A Petri net $N = \langle P, T, F \rangle$ is a workflow net (WF-net) if and only if P is a finite set of places, T is a finite set of transitions, $F \subseteq (P \times T) \cup (T \times P)$ is a finite set of arcs, $P \cap T = \emptyset$, and N has two special places i and o, where i has no predecessor and o has no successor.*

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 ${}^\bullet G = \cup_{g \in G}\ {}^\bullet g$. The state of a WF-net $N = \langle P, T, F \rangle$ is given by a marking function $M : P \to \mathbb{N}$ that associates a number of tokens to places. 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 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. Let $\sigma = t_1, t_2, .., t_{n-1}$ be a sequence of transitions of a Petri net N, $M_1 \xrightarrow{\sigma} M_n$ denotes that $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} .. \xrightarrow{t_{n-1}} M_n$. 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 $M_i \xrightarrow{\sigma} M_o$.

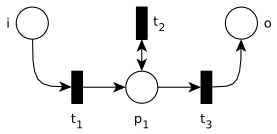

Fig. 1. An example of a WF-net

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 addition to *ordinary* WF-nets [4] (i.e. WF-nets with arcs of weight 1), this paper deals also with the following well-known and popular WF-net classes whose expressiveness is based on structural features:

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

2.2 Modal Specifications

Modal specifications have been designed to allow *loose* specifications to be expressed by imposing restrictions on transitions. They allow specifiers to indicate that a transition is *necessary* or just *admissible*. In [1], 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. Any modal specification formula $m \in S$ can be interpreted as a *may*-formula or a *must*-formula. A *may*-formula describes a behaviour that has to be ensured by at least one correct execution of the WF-net whereas a *must*-formula describes a behaviour that has to be ensured by all the correct executions of the WF-net. 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.

2.3 Modal Specifications Verification Method

This section provides an overall description of the verification method introduced in [1] to verify modal specifications of workflow nets. This method, based on the resolution of constraint systems, serves as a basis to compare SMT and CLP resolution approaches. 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 take its value from its domain. Such a system defines a Constraint Satisfaction Problem (CSP). Formally, a CSP 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), and C is a set of constraints $\{c_1(X_1), \ldots, c_m(X_m)\}$, where a constraint c_j involves 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 v (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, v is consistent—or satisfies a constraint $c(X)$ of C—if the projection of v on X is in $c(X)$. If v satisfies all the constraints of C, then Ω is a consistent or satisfiable CSP. In the rest of the paper, the predicate $SAT(C, v)$ is true if the corresponding CSP Ω is made satisfiable by v, and the predicate $UNSAT(C)$ is true if there exists no such v.

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 $N = (P, T, F)$, this method first models all the correct executions leading from M_a to M_b, i.e. all σ such that $M_a \xrightarrow{\sigma} M_b$. To reach that, the following constraint systems are defined:

Definition 2 (Minimum places potential constraint system). *Let $N = \langle P, T, F\rangle$ be a WF-net and M_a, M_b two markings of N, the minimum places potential constraint system $\varphi(N, M_a, M_b)$ associated with it is ($\nu : P \cup T \to \mathbb{N}$ defines a valuation function):*

$$\forall p \in P.\nu(p) = \sum_{t \in p^\bullet} \nu(t) + M_b(p) = \sum_{t \in {}^\bullet p} \nu(t) + M_a(p) \tag{1}$$

The solution's space of the constraint system from Eq. (1), Def. 2, defines an over-approximation of the executions of a workflow net based on the well-known state-equation [5]: If $M_a \xrightarrow{*} M_b$ then a valuation satisfying $\varphi(N, M_a, M_b)$ exists.

Definition 3. *Let $\theta(N)$ be the following constraint system associated with a WF-net $N = \langle P, T, F\rangle$ ($\xi : P \to \{0, 1\}$ defines a valuation function):*

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

This second constraint system from Def. 3 allows concluding on the existence of a siphon—an important structural feature describing a set of places $G \subseteq P$ such that $G \neq \emptyset$ and ${}^\bullet G \subseteq G^\bullet$—in a WF-net: N contains a siphon if and only if there is a valuation satisfying $\theta(N)$. Given a solution of the constraint system in Def. 2, it is then possible to build a subnet composed of the places (excluding places i and o) and of the transitions of the modelled execution as described in Def. 4.

Definition 4. *Let $N = \langle P, T, F\rangle$ be a WF-net, M_a, M_b two markings of N, and $\nu : P \cup T \to \mathbb{N}$ a valuation satisfying $\varphi(P, M_a, M_b)$. The subnet $sN(\nu)$ is defined as $\langle sP, sT, sF\rangle$ where:*

- $sP = \{p \in P \setminus \{i, o\} \mid \nu(p) > 0\}$
- $sT = \{t \in T \mid \nu(t) > 0\}$
- $sF = \{(a, b) \in F \mid a \in (sP \cup sT) \wedge b \in (sP \cup sT)\}$

By Th. 1, the above constraint systems can be combined to model executions of a workflow net.

Theorem 1. *Let $N = \langle P, T, F\rangle$ be a WF-net, and M_a, M_b its two markings. If there is $\nu : P \cup T \to \mathbb{N}$ such that $SAT(\varphi(N, M_a, M_b), \nu) \wedge UNSAT(\theta(sN(\nu)))$ $\wedge \forall n \in P \cup T.\ \nu(n) \leq 1$ then $M_a \xrightarrow{\sigma} M_b$ and $\forall t \in T.\ O_t(\sigma) = \nu(t)$.*

An execution modelled by the constraint system of Th. 1 is called a segment. Further, any execution of a workflow net can be modelled by a succession of segments as stated by Th. 2, in which the constraint system is denoted $\phi(N, M_a, M_b, k)$, where k is the number of segments composing the execution.

Theorem 2. *Let $N = \langle P, T, F\rangle$ be a WF-net, and M_a, M_b its two markings. $M_a \xrightarrow{\sigma} M_b$ if and only if there exists $k \in \mathbb{N}$ such that $M_1 \xrightarrow{\sigma_1} M_2 \cdots M_k \xrightarrow{\sigma_{(k)}}$ M_{k+1}, where $M_1 = M_a$, $M_{k+1} = M_b$ and for every $i,\ 0 < i \leq k$, there is ν_i s.t. $SAT(\varphi(N, M_i, M_{i+1}), \nu_i) \wedge UNSAT(\theta(sN(\nu_i))) \wedge \forall n \in P \times T.\ \nu_i(n) \leq 1$.*

Our method to verify modal specifications 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.

Theorem 3. *Let $N = \langle P, T, F\rangle$ be a WF-net and $\langle m, M\rangle$ a modal specification. The WF-net N satisfies the modal specification $\langle m, M\rangle$ if and only if:*

- *there is no $\nu, k \in \mathbb{N}$ such that $SAT(\phi(N, M_i, M_o, k) \wedge \neg C(m, \nu), \nu)$, and*
- *for every $f \in M$, there exist $\nu, k \in \mathbb{N}$ such that $SAT(\phi(N, M_i, M_o, k) \wedge C(f, \nu), \nu)$.*

By Th. 3, using a fixed K, the *K-bounded validity of a modal formula* (i.e. 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* [1].

3 Experimental Protocol

This section introduces the experimental protocol designed to evaluate the efficiency and limitations of the compared resolution approaches, i.e. CLP and SMT, applied to the verification of modal specifications as described in Sect. 2.3.

To empirically assess the scalability and the efficiency of both resolution methods, and to be able to have convincing clues to compare them as objectively as possible, we are interested in gathering the following abilities of the methods:

1. To assign a verdict about the (in)validity of the given modal specification;
2. To return such a response as quick as possible (and in an admissible time).

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 size by evaluating workflow nets of size up to 500 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. 2.1). Therefore, to experimentally evaluate both resolution approaches 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:
  - $50 * i$ where $i \in \{1, .., 10\}$
- 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 320 instances of growing size and complexity. All the evaluations have been performed on three different data sets, i.e. a total of 960 workflow nets and modal specifications have thus been experimented. Moreover, in order to restrict the total time needed to perform these experiments, a time-out of 10 minutes (arbitrary *admissible* time) was fixed for each resolution call to the solvers. Finally, all the executions have been computed on a computer featuring an Intel(R) Xeon(R) CPU X5650 @ 2.67GHz.

A complete and mature toolchain able to carry out this experimental protocol has been developed. To evaluate the constraint systems produced by the method in Sect. 2.3, this toolchain relies on either Z3 [6] version 4.4.0, an SMT solver, which finished first during the 2014 SMT-COMP challenge[1] for solving non-linear arithmetic problems, or SICStus Prolog [7] version 4.3.2, a CLP solver which obtained the third place during the 2014 MiniZinc challenge[2].

Figures 2 and 3 illustrate the encoding used to model the constraints seen in Sect. 2.3 respectively for the SMT-Lib and Prolog language. The constraints correspond with the workflow $N = (P, T, F)$ depicted in Fig. 1. Let $n \in P \cup T$, we use the following conventions: `An` stands for $M_a(n)$, `Bn` for $M_b(n)$, `Pn` for $\nu(n)$ when n is a place, `Tn` stands for $\nu(n)$ when n is a transition, and `Xn` stands for $\xi(n)$, `M1n`, `M2n` and `M3n` stand for $M_1(n)$, $M_2(n)$ and $M_3(n)$. For example, `Tt2` and `M2p1` respectively denotes $\nu(t2)$ and $M_2(p1)$.

```
; Variables declaration here
(define-fun initialMarking ((Ai Int)(Ao Int)(Ap1 Int)) Bool (and (= Ai 1) (= Ao 0) (= Ap1 0)))
(define-fun finalMarking ((Bi Int)(Bo Int)(Bp1 Int)) Bool (and (= Bi 0) (= Bo 1) (= Bp1 0)))
(define-fun stateEquation (
  (Ai Int)(Ao Int)(Ap1 Int)(Bi Int)(Bo Int)(Bp1 Int)(Pi Int)(Po Int)(Pp1 Int)(Tt1 Int)(Tt2 Int)(Tt3 Int)) Bool
  (and
    (>= Ai 0) (>= Ao 0) (>= Ap1 0) (>= Bi 0) (>= Bo 0) (>= Bp1 0)
    (>= Pi 0) (>= Po 0) (>= Pp1 0) (>= Tt1 0) (>= Tt2 0) (>= Tt3 0)
    (= Pi Ai) (= Pi (+ Bi Tt1))
    (= Po (+ Ao Tt3)) (= Po Bo)
    (= Pp1 (+ Ap1 Tt1 Tt2)) (= Pp1 (+ Bp1 Tt2 Tt3))))
(define-fun formula ((Tt1 Int) (Tt2 Int)) Bool (and (> Tt1 0) (> Tt2 0)))
(define-fun noSiphon (
  (Ai Int)(Ao Int)(Ap1 Int)(Bi Int)(Bo Int)(Bp1 Int)(Pi Int)(Po Int)(Pp1 Int)(Tt1 Int)(Tt2 Int)(Tt3 Int)) Bool
  (not (exists ((Xi Int)(Xo Int)(Xp1 Int))
    (and
      (> (+ Xi Xo Xp1) 0)
      (>= Xi 0) (<= Xi 1) (>= Xo 0) (<= Xo 1) (>= Xp1 0) (<= Xp1 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))
      (=> (> Tt1 0) (>= (+ Xi) Xp1)) (=> (> Tt2 0) (>= (+ Xp1) Xp1)) (=> (> Tt3 0) (>= (+ Xp1) Xo))))))
(define-fun segment (
  (Ai Int)(Ao Int)(Ap1 Int)(Bi Int)(Bo Int)(Bp1 Int)(Pi Int)(Po Int)(Pp1 Int)(Tt1 Int)(Tt2 Int)(Tt3 Int)) Bool
  (and
    (stateEquation Ai Ao Ap1 Bi Bo Bp1 Pi Po Pp1 Tt1 Tt2 Tt3)
    (noSiphon Ai Ao Ap1 Bi Bo Bp1 Pi Po Pp1 Tt1 Tt2 Tt3)))
```

Fig. 2. SMT-Lib code of a segment of workflow

```
initialMarking([1, 0, 0]).
finalMarking([0, 1, 0]).
stateEquation([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Tt1, Tt2, Tt3]):-
  domain([Ai, Ao, Ap1, Bi, Bo, Bp1, Pi, Po, Pp1, Tt1, Tt2, Tt3], 0, 10),
  Pi #= Ai, Pi #= Bi + Tt1,
  Po #= Ao + Tt3, Po #= Bo,
  Pp1 #= Ap1 + Tt1 + Tt2, Pp1 #= Bp1 + Tt2 + Tt3.
formula([Tt1, Tt2]):- Tt1 #> 0, Tt2 #> 0.
subnetInit([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Xi, Xo, Xp1]):-
  domain([Xi, Xo, Xp1], 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.
siphon([Tt1, Tt2, Tt3], [Xi, Xo, Xp1]):-
  Xi + Xo + Xp1 #> 0,
  Tt1 #> 0 #=> Xi #>= Xp1, Tt2 #> 0 #=> Xp1 #>= Xp1, Tt3 #> 0 #=> Xp1 #>= Xo,
  labeling([leftmost, step, up], [Xi, Xo, Xp1]).
noSiphon([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Tt1, Tt2, Tt3]):-
  subnetInit([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Xi, Xo, Xp1]),
  labeling([leftmost, step, up], [Tt1, Tt2, Tt3]),
  \+ siphon([Tt1, Tt2, Tt3], [Xi, Xo, Xp1]).
segment([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Tt1, Tt2, Tt3]):-
  stateEquation([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Tt1, Tt2, Tt3]),
  noSiphon([Ai, Ao, Ap1], [Bi, Bo, Bp1], [Pi, Po, Pp1], [Tt1, Tt2, Tt3]).
```

Fig. 3. Prolog code of a segment of workflow

[1] `www.smtcomp.org`

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

The two next inputs allow to determine, using respectively Z3 and SICStus, whether there exists a correct execution of the workflow in Fig. 1 made of three segments such that both `t1` and `t2` are fired.

Z3 input:

```
(assert (initialMarking M1i M1o M1p1))
(assert (finalMarking M3i M3o M3p1))
(assert (formula (+ T1t1 T2t1 T3t1) (+ T1t2 T2t2 T3t2)))
(assert (segment M1i M1o M1p1 M2i M2o M2p1 P1i P1o P1p1 T1t1 T1t2 T1t3))
(assert (segment M2i M2o M2p1 M3i M3o M3p1 P2i P2o P2p1 T2t1 T2t2 T2t3))
(assert (segment M3i M3o M3p1 M4i M4o M4p1 P3i P3o P3p1 T3t1 T3t2 T3t3))
(check-sat-using smt)
(get-model)
```

SICStus input:

```
initialMarking([M1i, M1o, M1p1]),
finalMarking([M4i, M4o, M4p1]),
segment([M1i, M1o, M1p1], [M2i, M2o, M2p1], [P1i, P1o, P1p1], [T1t1, T1t2, T1t3]),
segment([M2i, M2o, M2p1], [M3i, M3o, M3p1], [P2i, P2o, P2p1], [T2t1, T2t2, T2t3]),
segment([M3i, M3o, M3p1], [M4i, M4o, M4p1], [P3i, P3o, P3p1], [T3t1, T3t2, T3t3]),
S1 #= T1t1 + T1t2, S2 #= T2t1 + T2t2, formula([S1, S2]).
```

Both solvers give the following interpretation for the three segments:

```
M1i = 1, M1o = 0, M1p1 = 0, M2i = 0, M2o = 0, M2p1 = 1, M3i = 0, M3o = 0, M3p1 = 1, M4i = 0, M4o = 1, M4p1 = 0,
P1i = 1, P1o = 0, P1p1 = 1, P2i = 0, P2o = 0, P2p1 = 2, P3i = 0, P3o = 1, P3p1 = 1,
T1t1 = 1, T1t2 = 0, T1t3 = 0, T2t1 = 0, T2t2 = 1, T2t3 = 0, 3t1 = 0,
T3t2 = 0, T3t3 = 1
```

These segments are given in Fig. 4, starting (resp. ending) in the initial (resp. final) marking where only the input place `i` (resp. output place `o`) is marked.

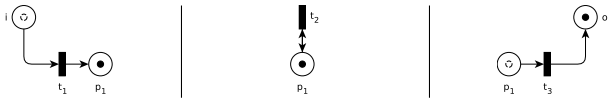

Fig. 4. The three segments of execution proposed by both solvers

Finally, let us precise that multiple combinations of labelling heuristics have been experimented when using SICStus. Though some of them may marginally improve the results on specific workflows, none was found to significantly and generally improve all results. Therefore, all the experiments have been conducted with the default options (i.e. [`leftmost`, `step`, `up`]). However, when using Z3, since the SMT tactic improved all results, this strategy has always been used.

4 Results and Feedback from Experiments

This section presents the experimental results obtained using the dedicated tool applying the protocol introduced in the previous section. 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;
- May-invalid and must-valid specifications.

Note that the algorithm given in Sect. 2.3 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. 2.3, 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 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 given 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 an discussed separately. Finally, for clarity, all time-outs and singularities have been withdrawn from the plots but systematically taken into account in our feedbacks. The interested reader can also study the complete data sets and results given at `https://dx.doi.org/10.6084/m9.figshare.2067156.v1`.

Tables 1, 2, 3 and 4 summarize the average verification times, number of time-outs as well as the overall appreciation of the results obtained over the different studied workflow net classes. Figures 5, 6, 7, 8, 9, 10, 11 and 12 depict the plots displaying the verification times spent by SICStus and Z3 for each class of workflow nets and types of modal specifications.

The next subsections comment on these obtained results and indicate the most important feedback for each class of workflow nets.

4.1 Observation from State-Machine Workflow Nets Verification

May-Valid and Must-Invalid specifications. Both solvers were able to conclude in a comparable and reasonable time. On average, Z3 execution time was 332ms whereas SICStus execution time was 704ms. However, despite good results for both solvers, it should be noted that 49.2%(59/120) of SICStus executions did not finish within 10 minutes, while Z3 did not suffer from any time-outs.

Table 1. Metrics over State-Machine workflow nets

Type	Solver	Avg. t.(ms)	#time-outs	Overall
May-Valid	Z3	346	0	🙂
	SICStus	621	28	😐
Must-Invalid	Z3	319	0	🙂
	SICStus	788	31	😡
May-Invalid	Z3	79	0	🙂
	SICStus	77413	52	😡
Must-Valid	Z3	79	0	🙂
	SICStus	10194	51	😡

Table 2. Metrics over Marked-Graph workflow nets

Type	Solver	Avg. t.(ms)	#time-outs	Overall
May-Valid	Z3	630	0	🙂
	SICStus	776	0	🙂
Must-Invalid	Z3	641	0	🙂
	SICStus	758	0	🙂
May-Invalid	Z3	112	0	🙂
	SICStus	424	0	🙂
Must-Valid	Z3	104	0	🙂
	SICStus	407	0	🙂

Table 3. Metrics over Free-Choice workflow nets

Type	Solver	Avg. t.(ms)	#time-outs	Overall
May-Valid	Z3	379	0	🙂
	SICStus	787	16	😐
Must-Invalid	Z3	413	0	🙂
	SICStus	898	14	😐
May-Invalid	Z3	91	0	🙂
	SICStus	40459	38	😡
Must-Valid	Z3	89	0	🙂
	SICStus	50566	37	😡

Table 4. Metrics over Ordinary workflow nets

Type	Solver	Avg. t.(ms)	#time-outs	Overall
May-Valid	Z3	1258	22	😐
	SICStus	9010	33	😡
Must-Invalid	Z3	713	17	😐
	SICStus	12258	37	😡
May-Invalid	Z3	108	0	🙂
	SICStus	9489	33	😡
Must-Valid	Z3	106	0	🙂
	SICStus	5949	37	😡

🙂: Reasonable time, no time-out — 😐: Reasonable time, # time-outs $< 50\%$ — 😡: # time-outs $> 50\%$

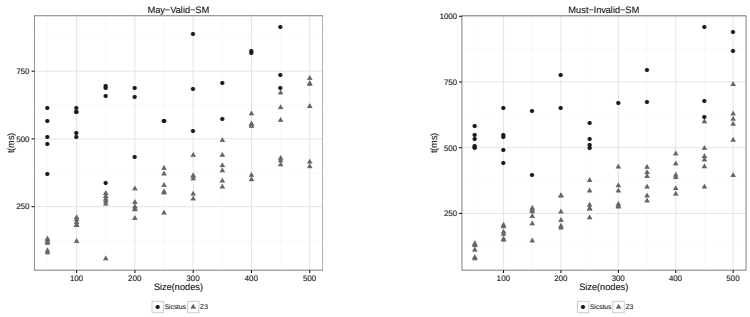

Fig. 5. State-Machine - May-Valid and Must-Invalid modal specifications

Must-valid and May-Invalid specifications. Both solvers were able to conclude in a reasonable time. On average, Z3 execution time was 79ms whereas SICStus execution time was 43803ms. These results clearly show that Z3 performs better than SICStus on this type of modal specifications. Moreover, it should be noted that 85.8%(103/120) of SICStus executions did not finish within 10 minutes, whereas Z3 did not suffer from any time-outs. Indeed, SICStus was not able to conclude about workflow nets of size greater than 250.

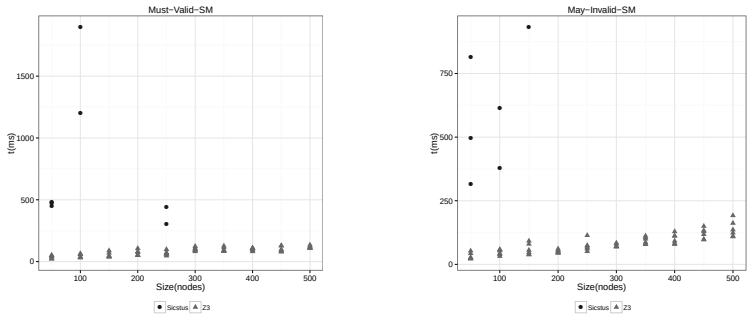

Fig. 6. State-Machine - Must-Valid and May-Invalid modal specifications

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

4.2 Observation from Marked-Graph Workflow Nets Verification

May-Valid and Must-Invalid specifications. Both solvers were able to conclude in a reasonable and comparable time. On average, Z3 execution time was 635ms whereas SICStus execution time was 767ms. It should also be pointed out that for large sized Marked-Graph workflow nets (greater than 400 nodes) SICStus performs slightly better than Z3.

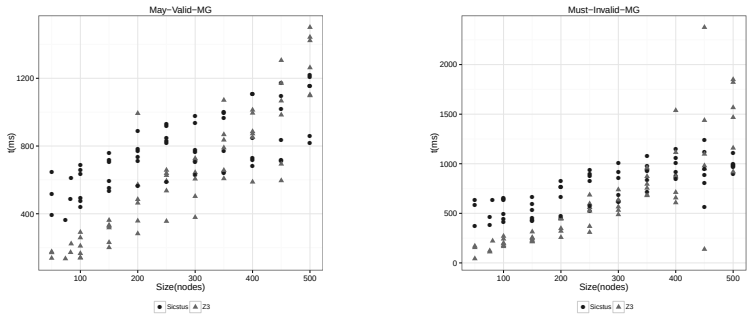

Fig. 7. Marked-Graph - May-Valid and Must-Invalid modal specifications

Must-valid and May-Invalid specifications. Both solvers were able to conclude in a reasonable and fairly comparable time. On average, Z3 execution time was 108ms whereas SICStus execution time was 415ms. Besides, notice that, for this type of modal specifications, Z3 performs slightly better than SICStus.

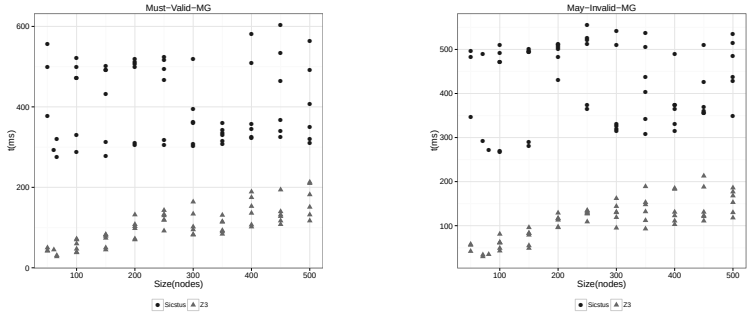

Fig. 8. Marked-Graph - Must-Valid and May-Invalid modal specifications

Synthesis. Over the class of Marked-Graph, we can conclude that SICStus and Z3 performs similarly. However SICStus seems to perform better when verifying May-Valid and Must-Invalid specifications while Z3 seems to perform better when verifying Must-valid and May-Invalid specifications. A further investigation has shown that, in general, Z3 is more effective than SICStus for the computation of the over-approximation used by the verification method, while SICStus is more effective than Z3 for the computation of the segments needed to conclude whenever the over-approximation is not sufficient.

4.3 Observation from Free-Choice Workflow Nets Verification

May-Valid and Must-Invalid specifications. Both solvers were able to conclude in a reasonable and comparable time. On average, Z3 execution time was 396ms whereas SICStus execution time was 842ms. Beyond these conclusive results for both solvers, it is important to underline that 25%(30/120) of SICStus executions did not finish within 10 minutes, whereas Z3 did not suffer from any time-outs.

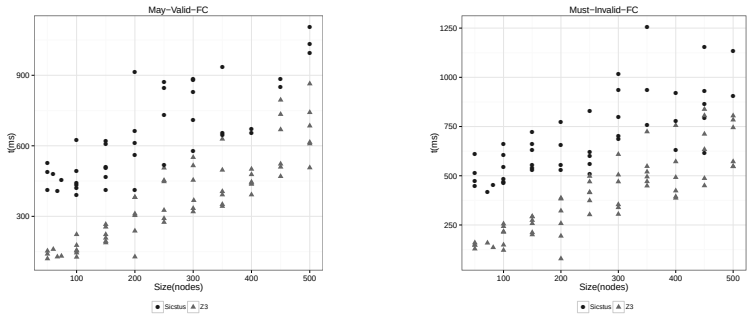

Fig. 9. Free-Choice - May-Valid and Must-Invalid modal specifications

Must-valid and May-Invalid specifications. Over this type of modal specifications, Z3 clearly performs better than SICStus. On average, Z3 execution time was 90ms, while SICStus execution time was 45512ms. We also note that 62.5%(75/120) of SICStus executions did not finish within 10 minutes, whereas Z3 did not suffer from any time-outs. 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 far better off.

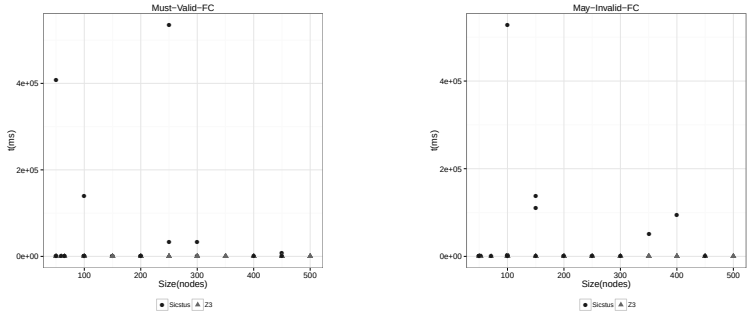

Fig. 10. Free-Choice - Must-Valid and May-Invalid modal specifications

Synthesis. Over the class of Free-Choice workflow nets, we observe that Z3 performs better than SICStus. 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.

4.4 Observation from Ordinary Workflow Nets Verification

May-Valid and Must-Invalid specifications. Both solvers were able to conclude in a reasonable and comparable time. On average, Z3 execution time was 985ms whereas SICStus execution time was 10634ms. Besides these results, it should be underlined that 58.3%(70/120) of SICStus executions and that 32.5%(39/120) of Z3 executions did not finish within 10 minutes.

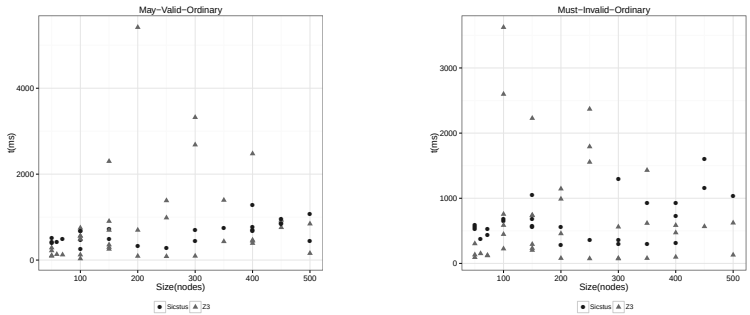

Fig. 11. Ordinary - May-Valid and Must-Invalid modal specifications

Must-valid and May-Invalid specifications. Both solvers were able to conclude in a reasonable time. On average, Z3 execution time was 107ms whereas SICStus execution time was 7717ms. Despite these conclusive results for both solvers, it is important to note that 58.3%(70/120) of SICStus executions did not finish within 10 minutes, whereas Z3 did not suffer from any time-outs. As for the previous classes, Z3 indeed performs better than SICStus to compute the over-approximation constraints, which were often sufficient to conclude.

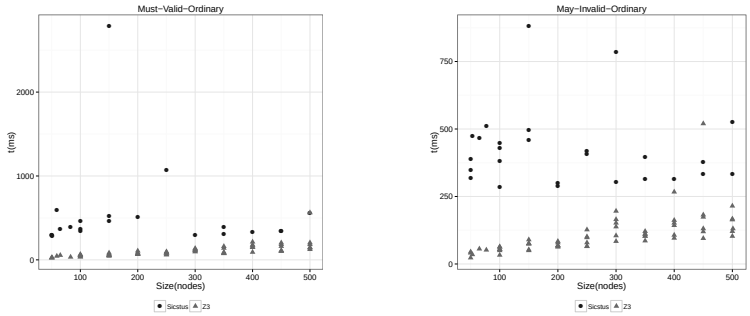

Fig. 12. Ordinary - Must-Valid and May-Invalid modal specifications

Synthesis. Over the class of ordinary workflow nets, we observe that Z3 performs better than SICStus, especially when verifying Must-valid and May-Invalid specifications. We can thus conclude from these results that the SMT approach seems to be more suited for the modal specifications verification over ordinary workflow nets. The next section summarizes the lessons learned and the benefits noticed from these experiments according to the initial challenges.

4.5 Lessons Learned from Experience

Scalability. On the basis of the results, we can confidently state that the verification method proposed in [1] is scalable in terms of modal specification and workflow net complexity, as well as regarding their size (up to at least 500 nodes).

Efficiency. The developed implementation of the method proposed in [1] and the underlying constraint solvers (i.e. Z3 and SICStus) have shown to be very efficient for the intended verification computation. Indeed, the toolchain was always able to conclude about the validity of modal specification over workflow nets of growing size and complexity within the alloted time of 10 minutes (for each constraint system to solve, at least one resolution method was indeed able to assign a verdict, and furthermore within only few seconds in almost all cases). Furthermore, we observed that memory usage does not seem to be a limiting factor because, for the biggest instance of workflow verification, the memory usage for SICStus and Z3 was less than 350MB.

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. We indeed 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.

5 Related Work and Conclusion

On workflow nets verification. Verifying properties over business processes has been widely investigated using Petri-net-based approaches. Among them, workflow nets constitute a suited class for modelling business process [4]. Thus, approaches and tools [1, 8, 9] have emerged to verify properties over these workflow nets and, as a consequence, over the processes they model. However, regarding verification of such WF-nets, the reachability problem, proved to be an EXPSPACE problem in [10], is the key problem that all approaches are facing.

Regarding verification methods, some research results have also been proposed to express and verify properties against a given system. Let us quote [11] where the expression of properties with modalities is investigated for automata/-transition systems, and also [12] 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.

Using constraint solving for verification. 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 [13] for checking the reachability of bounded Petri nets, as well as in [14] 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 [15] as well as Petri nets [9]. In a very similar way, a CLP approach has been used in [16] 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.

Conclusion. This paper has compared the SMT and CLP approaches to verify modal specifications over WF-nets using constraint solving. For this purpose, an experimental protocol has been designed and a mature toolchain has been developed. Using the obtained experimental results over four classes of nets with particular features, we have empirically demonstrated that the verification method is efficient and scalable over workflow nets of size at least up to 500 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. As a future work, we plan to apply our approach to real-life industrial workflows to confirm its efficiency, and to investigate innovative 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.

References

1. Bride, H., Kouchnarenko, O., Peureux, F.: Verifying modal workflow specifications using constraint solving. In: Proc. of Integrated Formal Methods (IFM'14). Volume 8739 of LNCS., Bertinoro, Italy, Springer (September 2014) 171–186
2. Tsang, E.P.K.: Foundations of constraint satisfaction. Computation in cognitive science. Academic Press (1993)
3. De Moura, L., Bjørner, N.: Satisfiability modulo theories: Introduction and applications. Commun. ACM **54**(9) (September 2011) 69–77
4. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers **08**(01) (1998) 21–66
5. Murata, T.: Petri nets: Properties, analysis and applications. IEEE **77**(4) (April 1989) 541–580
6. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (2008) 337–340
7. Carlsson, M., et al.: SICStus Prolog user's manual (Release 4.2.3), Swedish Institute of Computer Science, Kista, Sweden. (October 2012)
8. Bi, H., Zhao, J.: Applying propositional logic to workflow verification. Information Technology and Management **5**(3-4) (2004) 293–318
9. Wimmel, H., Wolf, K.: Applying CEGAR to the Petri net state equation. Logical Methods in Computer Science **8**(3) (2012)
10. Haddad, S.: Decidability and complexity of Petri net problems. Petri Nets: Fundamental Models, Verification and Applications (2009) 87–122
11. Larsen, K., Thomsen, B.: A modal process logic. In: Logic in Computer Science, 1988. LICS '88., Proc. of the Third Annual Symposium on. (1988) 203–210
12. Kouchnarenko, O., Sidorova, N., Trcka, N.: Petri Nets with May/Must Semantics. In: Wsh. on Concurrency, Specification, and Programming - CS&P 2009. Volume 1., Kraków-Przegorzaly, Poland (September 2009)
13. Monakova, G., Kopp, O., Leymann, F., Moser, S., Schäfers, K.: Verifying business rules using an SMT solver for BPEL processes. In: BPSC. Volume 147 of LNI., GI (2009) 81–94
14. Pólrola, A., Cybula, P., Meski, A.: Smt-based reachability checking for bounded time Petri nets. Fundam. Inform. **135**(4) (2014) 467–482
15. Kleine, M., Göthel, T.: Specification, verification and implementation of business processes using CSP. In: TASE, IEEE Computer Society (2010) 145–154
16. Soliman, S.: Finding minimal p/t-invariants as a csp. In: Proc. of the 4th Wsh. on Constraint Based Methods for Bioinformatics WCB. Volume 8. (2008)