A Survey of Formal Business Process Verification  
From Soundness to Variability

Heerko Groefsema and Doina Bucur
Johann Bernoulli Institute for Mathematics and Computer Science,  
University of Groningen, Nijenborgh 9, 9747 AG Groningen, the Netherlands  
{h.groefsema, d.bucur}@rug.nl

Keywords: Business Process Management, Verification, Model Checking, Survey.

Abstract: Formal verification of business process models is of interest to a number of application areas, including checking for basic process correctness, business compliance, and process variability. A large amount of work on these topics exist, while a comprehensive overview of the field and its directions is lacking. We provide an overview and critical reflections on existing approaches.

1 INTRODUCTION

Business process modeling helps businesses to increase the quality of their processes. Formal techniques are used to model, implement, execute, and monitor business process models. Model checking is a technique which verifies a given system model for compliance with a specification of interest, for various practical goals including ensuring basic correctness of processes, business compliance checking, and process variability. A related survey (Mori-moto, 2008) provides an overview of business process checking, but does not consider compliance and variability as supported through formal verification, while in (Aiello et al., 2010), we survey variability for business processes.

While a large amount of work exists in the field of business process verification, it lacks an overview of the state of the field and its related formal verification frameworks. As such, in the present treatment, we aim to provide an overview of formal verification goals, techniques, and frameworks for business process modeling, and give critical reflections.

Frameworks aiming at the verification of business process models exist. They supports various process specification formalisms, e.g., imperative, declarative, event-driven, or artifact-centric. In imperative specification formalisms, processes are modelled as sets of tasks or activities, gates, and events interlinked by flows or transitions. Each activity describes a single unit of work and the transitions describe the order between these units of work. Common notations include Business Process Model and Notation (BPMN) (OMG, 2011), Business Process Execution Language (BPEL) (Oasis, 2007), Unified Modeling Language (UML) activity diagrams (OMG, 2011), and Yet Another Workflow Language (YAWL) (Hof-stede et al., 2010). Alternatively, declarative specification formalisms, such as (Pesic and van der Aalst, 2006), model processes without distinct flow controls which specify order between units of work. Instead, these specifications express a process model as a set of activities and a set of constraints over these activities, with the constraints restricting the possible inclusion and ordering of the activities. Any process behaviour not prohibited by these constraints is valid.

Event-driven specification is another approach to business process modelling. Defined in (Keller et al., 1992), Event-driven Process Chains (EPC) are directed graphs mainly consisting of events, functions (activities), and logical connectors (gates). Unlike imperative specifications, EPC do not model node ordering explicitly. Although EPC are known for their intelligible notation and simplicity, their lack of semantics is a topic of discussion (van der Aalst, 1999). Finally, artifact-centric specifications focus on the evolution of business entities and data. Originally proposed in (Nigam and Caswell, 2003), such specifications incorporate the notion of the lifecycle of business artifacts, such as data—which is ignored by most other specifications.

In what follows, Section 2 examines the techniques and goals of business processes verification. We classify and discuss verification frameworks in Section 3, and conclude in Section 4.
2 BUSINESS PROCESS MODEL CHECKING

Business processes verification can be based on several algorithmic techniques (and some supporting intermediary modelling formalisms), and be used for a number of goals. Next we overview these.

2.1 Verification Techniques

Petri Nets are state-transition systems used to analyze distributed systems. They are amenable to intuitive graphical notation which, unlike most of the process notations discussed in Section 1, include a mathematical definition of its execution semantics. A number of subclasses of Petri Nets have been defined, most notably the sub-class of Workflow (WF) nets (van der Aalst, 1998) which is applied as an intermediary formalism in the verification of business processes. Besides Petri Nets, a system may also be modelled as a finite state machine (FSM)—a directed graph of nodes and edges, with nodes representing a system state system and edges representing a change in state. Both Petri Nets and FSM-like models are then verified against a specification or correctness property.

On the other hand, generic model checkers, e.g., SPIN (Holzmann, 2004) and nuSMV2 (Cimatti et al., 1999), implement search algorithms to verify any system modelled in the model checker’s input languages. Of these, notable is the Process Meta-Language, or Promela, used by SPIN. Business processes can be remodelled using Promela, with the Promela implementation then internally translated into an automaton and verified against a correctness property.

Correctness properties may be informal specifications, process models, or logic formulas. Informal specifications include properties defined as simple tuples or programming methods. Process models themselves can be used as correctness properties as well. In this case, the original business process model is verified to be a refinement of the correctness model. Logic properties are formulas in logics ranging from propositional logic to deontic and temporal logic. Deontic logics reason about obligations and permissions. Temporal logics include Linear Temporal Logic (LTL) and Computation Tree Logic (CTL), a branching-time logic. LTL specifies properties (e.g., the universality of a certain state property, and the order of states) over states occurring on process execution paths. CTL extends this set of temporal operators with path quantifiers, such that formulas can specify properties over branching executions. Extensions of these logics are common for process verification. They include Past-time LTL (PLTL), but also novel logics for business process specification.

2.2 Goals of Verification

Business process verification is the act of determining if a business process model is correct with regard to a set of formal correctness properties. Often, verification is automated by tools known as analyzers or model checkers. Several goals for using verification are presented in the business process literature.

The first goal is verifying basic properties such as reachability and termination. Reachability of a business activity requires an execution path to exist leading to that activity starting from the initial activities. A termination property requires that all possible execution traces terminate. Business process soundness, a property originally proposed in the area of Petri Net verification, is known as the combination of these two properties plus a third: the absence of related running activities at process termination (i.e., proper completion). Avoiding the deployment of erroneous processes that do not conform with these properties is obviously advantageous: “[erroneously designed workflow models can result in failed workflow processes, execution errors, and disgruntled customers and employees” (Bi and Zhao, 2004).

The second goal for business process verification is business compliance. Compliance checks whether process models conform with specifications, which in this case can be another process model or a set of rules, such as (inter)national laws and standards. When verifying compliance, rules are often specified using a formal logic over the entities (e.g., events, activities) of the business process model. In other cases, these rules are informally specified. For example, regulations could specify that before processing a wire transfer, a bank should identify if any sanctions exist regarding the involved parties.

The third goal of verification of process models, variability, extends upon compliance. “In the context of BPM, variability indicates that parts of a business process remain variable, or not fully defined, in order to support different versions of the same process depending on the intended use or execution context” (Aiello et al., 2010). Variability aims to support different versions of the same process. This includes support of process families at design-time, when a new process variant can be derived from a generic process, and process flexibility or adaptability at runtime, where a generic process can be adapted. Variability can be specified in two different ways. The first, which is not in the scope of this survey, employs the use of variation points to provide different options at specific points in a process. The second, which is in
Table 1: Soundness tools and frameworks.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Modelling</th>
<th>Intermediate</th>
<th>Correctness properties</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>(van der Aalst, 1998)</td>
<td>Petri Net</td>
<td></td>
<td></td>
<td>Wollan</td>
</tr>
<tr>
<td>(Bi and Zhao, 2004)</td>
<td>WIMC</td>
<td></td>
<td>Propositional logic</td>
<td>Algorithmic</td>
</tr>
<tr>
<td>(Choi and Zhao, 2005)</td>
<td>WIMS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(van Dongen et al., 2007)</td>
<td>EPC</td>
<td>Petri Net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Fisteus et al., 2005)</td>
<td>BPEL/AWS</td>
<td>CFM</td>
<td>LTL, CTL</td>
<td>SPIN, SMV</td>
</tr>
<tr>
<td>(Karamanolis et al., 2000)</td>
<td>WIMS</td>
<td>FSP</td>
<td></td>
<td>LTSA</td>
</tr>
<tr>
<td>(Koehler et al., 2002)</td>
<td>Imperative</td>
<td>FSM</td>
<td>CTL</td>
<td>muSMV</td>
</tr>
<tr>
<td>(Masalagiu et al., 2009)</td>
<td>BPMN</td>
<td></td>
<td>LTL, CTL (via TLA+)</td>
<td>TLC</td>
</tr>
<tr>
<td>(Nakajima, 2006)</td>
<td>BPEL</td>
<td>EFA to Promela</td>
<td></td>
<td>SPIN</td>
</tr>
<tr>
<td>(Weber et al., 2010)</td>
<td>Annotated process</td>
<td>Annotated process</td>
<td></td>
<td>Algorithmic</td>
</tr>
<tr>
<td>(Wynn et al., 2009)</td>
<td>YAWL</td>
<td>Petri Net</td>
<td></td>
<td>YAWL</td>
</tr>
</tbody>
</table>

3 OVERVIEW OF FRAMEWORKS

The existing frameworks aim to verify business processes either at design time or at runtime. The most basic form of verification of processes deals with design-time verification of soundness properties, e.g., termination and reachability. Table 1 gives an overview of these frameworks. (van der Aalst, 1998) introduced soundness to the field of BPM by translating workflows into Petri Nets, and (Wynn et al., 2009) perfected the application by allowing Or-joins and cancelation regions. Due to this, Petri Nets are commonly used as intermediate formalisms by soundness verification frameworks, including (van Dongen et al., 2007), who use it to verify EPC. Another popular method is by translating processes into a model checker input language, e.g., (Masalagiu et al., 2009) verify BPMN by translating it (via a Petri Net intermediate model) into the model checker input language TLA+. (Karamanolis et al., 2000) translate processes to the process algebra FSP and checks the result with the Labeled Transition System Analyzer, (Koehler et al., 2002) translate into the nuSMV input language, and (Nakajima, 2006) translates into the SPIN language Promela.

Other frameworks verify business compliance; an overview of these is given in Table 2. A dominant number of compliance frameworks focus on verifying imperative specifications such as BPMN, BPEL, EPC, and UML sequence diagrams. (Anderson et al., 2005; Arbab et al., 2009; Awad et al., 2008; Foster et al., 2003; Ghose and Kohliadis, 2007; Goedertier and Vanthienen, 2006; Janssen et al., 1998; Liu et al., 2007; Ly et al., 2008; Ly et al., 2011; Nakajima, 2002) all belong in this category. Others extend declarative specifications with compliance features. In (Chesani et al., 2009), compliance is modelled based on DecSerFlow, a declarative runtime process specification, by translating it into a reactive event calculus. (Pulvermueller et al., 2010) aims at verifying the compliance of design-time EPC using an extension of CTL that differentiates between events and functions. Finally, (Deutsch et al., 2009) proposes verifying the compliance of artifact-centric processes against properties expressed in an extension of LTL.

The last set of frameworks (listed in Table 3) aim at supporting variability. Declarative variability extends upon compliance by only specifying rules over the set of tasks in a process, instead of building an imperative graph. As such, the authors of (Governatori et al., 2006) first propose a compliance framework based upon the newly proposed deontic logic FCL, then continue by extending this framework, in (Governatori et al., 2011), with goals to provide a fully declarative description. (Sadiq et al., 2005) proposes defining pockets of flexibility within imperatively specified processes to introduce design-time variability, using constraints which provide ordering and inclusion information but which are not specified using a formal logic. Formal frameworks base their declarative specifications on the temporal logics LTL and CTL. As examples, (Demeyer et al., 2010) use Finite LTL to specify fully declarative processes, (D’Aprile et al., 2011) specify declarative processes using temporal Answer Set Programming (ASP) and Dynamic LTL, (Pesci and van der Aalst, 2006) use LTL to specify flexible run-time processes, and the related work of (van der Aalst and Pesic, 2006) aims towards service flows instead. Finally, (Maggi et al., 2011) extends upon the work of (Pesic and van der
Table 2: Compliance tools and frameworks.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Modelling</th>
<th>Correctness properties</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Anderson et al., 2005)</td>
<td>UML Sequence Diagram</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Arbab et al., 2009)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Awad et al., 2008)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Chesani et al., 2009)</td>
<td>DecSerFlow</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Deutsch et al., 2009)</td>
<td>Business artifacts</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Foster et al., 2008)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Ghose and Koliadis, 2007)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Janssen et al., 1998)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Ly et al., 2008; Ly et al., 2011)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Montali et al., 2010)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Nakajima, 2002)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Pulvermueller et al., 2010)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
<tr>
<td>(Weber et al., 2008)</td>
<td>BPMN</td>
<td>CSP</td>
<td>FDR</td>
</tr>
</tbody>
</table>

Critically, drawbacks exist in certain frameworks for compliance and variability. Some frameworks inefficiently translate the business-process model into a model checker input language, introducing a large overhead in the ensuing state space (e.g., a simple process of five activities and four transitions is reportedly mapped to 201 states and 586 transitions in SPIN by (Nakajima, 2002)).

Other methods for design-time verification (but not for runtime verification, which checks linear execution paths) lack good support for complex branching features of the modelling formalism (e.g., do not support parallel gates, or execution loops). To improve on this, some introduce workarounds, e.g., (Pulvermueller et al., 2010) proposes using simple variables on automata to fork exclusive paths and synchronize parallel paths. In other work (Feja et al., 2009), the same authors propose an unsound Kripke translation when dealing with parallel paths. Here, pairs of activities from different parallel paths are explicitly synchronized; in reality, however, each path should only be synchronized after a join. Other frameworks, e.g., (Sadiq et al., 2005), (Choi and Zhao, 2005), and (Grofetsema et al., 2011), simply ignore exclusive, inclusive, and/or parallel paths for reasons of complexity. (Weber et al., 2008) proposes two verification algorithms in polynomial time; however, one is reported by the authors as unsound and complete, the other sound but incomplete, and both only support acyclic processes. (Montali et al., 2010) reports its verification algorithm to be unable to terminate under certain loops.

Finally, other frameworks require users to apply newly proposed or extended specification logics in order to specify correctness properties. Examples include (Pulvermueller et al., 2010), which extends CTL with the ability to differentiate between events and functions in EPC, (Governatori et al., 2006), which proposes an entirely new deontic logic, and (Bulanov et al., 2011), which proposes a new temporal process logic to allow process mergers.

4 CONCLUSIONS

Formal verification of business processes was initially proposed to check for process soundness, but lately has been deployed to also support business compliance and variability. While process soundness is well supported by frameworks based on Petri-Net formalisms, the areas of compliance and variability checking lack design-time solutions which (i) minimize the overhead of states in the formal model, and (ii) support large subsets of the business process modelling formalism, including parallel gates and process loops.
Table 3: Variability tools and frameworks.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Formalisms</th>
<th>Modeling</th>
<th>Intermediate</th>
<th>Correctness properties</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>(D’Aprile et al., 2011)</td>
<td>Imperative</td>
<td>TPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Bulanov et al., 2011)</td>
<td>BPBMN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Governatori et al., 2011)</td>
<td>BPMN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Groeneveld et al., 2011)</td>
<td>BPMN + CTL</td>
<td>CTL</td>
<td></td>
<td>VxBPMN</td>
<td></td>
</tr>
<tr>
<td>(Demeyer et al., 2010)</td>
<td>Saturn, Automata</td>
<td>Finite LTL</td>
<td></td>
<td>Saturn Eng.</td>
<td></td>
</tr>
<tr>
<td>(Pesic and van der Aalst, 2006; van der Aalst and Pesic, 2006; Maggi et al., 2011)</td>
<td>LTL</td>
<td>Automata</td>
<td>LTL</td>
<td>Declare</td>
<td></td>
</tr>
<tr>
<td>(Rychkova et al., 2008)</td>
<td>BPMN + FO</td>
<td>FO</td>
<td></td>
<td>FO</td>
<td></td>
</tr>
<tr>
<td>(Sadiq et al., 2005)</td>
<td>Informal</td>
<td>Chameleon</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REFERENCES


A Survey of Formal Business Process Verification - From Soundness to Variability


