Verification of Time-Aware Business Processes using Constrained Horn Clauses*

Emanuele De Angelis[1], Fabio Fioravanti[1], Maria Chiara Meo[1]
Alberto Pettorossi[2,3], and Maurizio Proietti[3]

[1] DEC, University 'G. D'Annunzio', Pescara, Italy
{emanuele.deangelis,fabio.fioravanti,cmeo}@unich.it
[2] DICII, University of Rome Tor Vergata, Rome, Italy
pettorossi@disp.uniroma2.it
[3] IASI-CNR, Rome, Italy
maurizio.proietti@iasi.cnr.it

Abstract. We present a method for verifying properties of time-aware business processes, that is, business processes where time constraints on the activities are explicitly taken into account. Business processes are specified using an extension of the Business Process Modeling Notation (BPMN) and durations are defined by constraints over integer numbers. The definition of the operational semantics is given by a set *OpSem* of constrained Horn clauses (CHCs). Our verification method consists of two steps. (Step 1) The specialization of *OpSem* with respect to a given business process and a given temporal property to be verified. This specialization produces a set of CHCs whose satisfiability is equivalent to the validity of the given property. (Step 2) The use of any state-of-the-art solver for CHCs to check the satisfiability of such sets of clauses. We have implemented our verification method using the VeriMAP transformation system and the Z3 solver for CHCs.

1 Introduction

A *business process*, or BP for short, consists of a set of activities, performed in coordination within a single organization, which realize a business goal [31,34]. The *Business Process Model and Notation*, or BPMN for short, is one of the most popular graphical languages proposed for visualizing business processes [27]. The primary goal of BPMN is to provide a standard notation that can be understood by all business stakeholders, which include the business analysts who define and modify the processes, the technical developers in charge of their implementation, and the business managers who monitor and manage the processes.

A BPMN model is a procedural, semi-formal description of the order of execution of the activities of a given process and how these activities must coordinate, abstracting away from many other aspects of the process itself, such as the manipulation of data and the duration of the activities. However, for many analysis tasks these aspects are very significant in practice and should be taken

* This work has been partially funded by INdAM-GNCS (Italy).

into consideration. In particular, the duration of the activities is crucial when we want to reason about time constraints (such as deadlines or earliest completion times) that should be satisfied by the executions of the process.

Various approaches for BP modeling with duration and time constraints have been proposed in the literature (see [6] for a recent survey). Some of these approaches define the semantics of *time-aware* BPMN models by means of formalisms such as *time Petri nets* [24], *timed automata* [32], and *process algebras* [35]. Properties of these models can then be verified by using very effective reasoning tools available for those formalisms [4,14,22].

However, the above mentioned formalisms and tools may not be adequate if we want to complement time-based reasoning with general purpose logical reasoning, which is often needed if we take into account more complex aspects of knowledge manipulation activities relative to business processes. For instance, some verification approaches make use of ontology-based reasoning about the business domain where processes are executed [30,33], while others combine reasoning on the finite-state process behavior with reasoning on the manipulation of data objects of infinite types such as databases or integers [2,9,29].

Thus, in view of an integration of various reasoning tasks needed to analyze business processes from different perspectives, we propose a logic-based approach to modeling and verifying time-aware business processes.

The main contributions of the paper are the following. We present a logic-based language to specify time-aware BPMN models, where time and duration of activities are explicitly represented. Then we define an operational semantics of time-aware BPMN models by means of deduction rules that allow us to infer the time intervals when a particular activity is in execution or 'is enacting', using the BPMN terminology. Next, in order to prove properties of time-aware BPMN models, we follow a transformational approach similar to the one proposed in [11] for the verification of imperative programs. First, we consider an encoding *OpSem* of the operational semantics of business processes into *Constrained Horn Clauses* (CHCs) [5] (or, equivalently, *Constraint Logic Programs* [20]). Then, we specialize *OpSem* with respect to the time-aware BPMN model under consideration and the temporal property of interest, thereby deriving a new set of CHCs whose satisfiability is equivalent to (and thus implies) the validity of the property to be verified. Finally, we use the state-of-the-art solver Z3 [12] for CHCs to check the satisfiability of such set of clauses.

Since the CHCs are generated in an automatic way by the CHC specializer from the formal definition of the semantics of the BPMN models, and the CHC solvers are general purpose reasoning systems, our approach is, to a large extent, parametric with respect to other extensions of BP models one may want to consider in the future. Moreover, recent advances in the field of CHC solving can be exploited to get very effective reasoning tools for verifying other classes of properties of business processes besides the temporal ones.

The paper is structured as follows. In Section 2 we recall some basic notions about Constrained Horn Clauses (CHCs) over integer numbers and Business Process Model and Notation (BPMN). In Section 3 we present our logic-based language for specifying time-aware BPMN models and the operational semantics of the language. In Section 4 we present the CHC encoding of the semantics and the transformation techniques for specializing *OpSem* with respect to a given time-aware BPMN model and a given property. In Section 5 we report on the implementation of the verification technique we have made using the VeriMAP transformation and verification system [10], and the CHC solver Z3. Finally, in Section 6 we discuss related work in the field of Business Process verification.

2 Preliminaries

In the next two subsections we recall some basic notions concerning constrained Horn clauses and the Business Process Model and Notation.

We consider time to be a discrete quantity and we consider the ‘time line’ to be the set of integers. However, our approach applies directly to dense or continuous time as well.

2.1 Constrained Horn Clauses over Integers

First we need the following notions about constraints, constrained Horn clauses, and constraint logic programming. For related notions not familiar to the reader, we refer to [20,23].

Constraints are defined as follows. Let *RelOp* be the set of predicate symbols $\{=, \neq, \leq, \geq, <, >\}$. If p_1 and p_2 are linear polynomials with integer variables and coefficients, then $p_1 R\, p_2$, with $R \in \mathit{RelOp}$, is an *atomic constraint.* A *constraint* c is a (possibly empty) conjunction of atomic constraints. An *atom* is a formula of the form $p(t_1, \ldots, t_m)$, where p is a predicate symbol not in *RelOp* and $t_1, \ldots, t_m$ are terms constructed as usual from variables, constants, and function symbols. In particular, we assume that there are two predicate symbols *true* and *false* of arity 0, and a predicate symbol *eq* denoting identity. A *constrained Horn clause* (or simply, a *clause*) is an implication of the form $A \leftarrow c, G$, where the conclusion (or *head*) A is an atom, and the premise (or *body*) ‘c, G’ is the conjunction of a constraint c and a (possibly empty) conjunction G of atoms. The empty conjunction is identified with *true.* A *constrained fact* is a clause of the form $A \leftarrow c$, and if c is *true* we will call it simply a *fact.* A *constrained goal* (or simply, a *goal*) is a clause of the form $\mathit{false} \leftarrow c, G$. Given a formula φ, $\mathit{vars}(\varphi)$ denotes the set of variables occurring in φ. A clause C is said to be *ground* if $\mathit{vars}(C) = \emptyset$.

Given a set $\mathcal{P}$ of clauses, $\mathbb{Z}$-*interpretation* is defined to be an interpretation I of $\mathcal{P}$ such that: (i) *true* holds in I, (ii) *false* does not hold in I, (iii) I is the usual interpretation over the set of the integer numbers $\mathbb{Z}$ for the constraints, and (iv) I is the Herbrand interpretation for predicate and function symbols not in $\mathit{RelOp} \cup \{\mathit{true}, \mathit{false}, +, \times\}$ (in particular, $\mathit{eq}(x, y)$ holds if and only if x and y are identical terms in the Herbrand universe). For any formula φ we write $\mathbb{Z} \models \varphi$ if φ holds in all $\mathbb{Z}$-interpretations. A $\mathbb{Z}$-*model* of $\mathcal{P}$ is a $\mathbb{Z}$-interpretation M such that every clause of $\mathcal{P}$ holds in M. A set of CHCs is *satisfiable* if it has a $\mathbb{Z}$-*model.* (Note that a set of CHCs may be unsatisfiable if it contains goals.) Every satisfiable set $\mathcal{P}$ of CHCs has a unique *least* $\mathbb{Z}$-*model*, denoted $M(\mathcal{P})$ [20].

2.2 Business Processes Model and Notation

A BPMN model is defined through a diagram drawn by using graphical constructs representing *flow objects* and *sequence flows* (sequence flows will also be called *flows* for short). That diagram can be extended, if so desired, to include information about data flow, resource allocation (for instance, how the work to be done is assigned to the participants in the process), and exception handling (for instance, how erroneous behaviors should be handled).

For reasons of simplicity, in this paper we will only consider a subset of the flow objects and sequence flows that can occur in a BPMN model, but our approach can easily be extended to full BPMN. The flow objects we will consider are of three kinds: either (i) *tasks*, denoted by rounded rectangles, or (ii) *events*, denoted by circles, or (iii) *gateways*, denoted by diamonds. Tasks represent atomic units of work performed within the process. Events denote something that happens during the execution, or the *enactment*, using the BPMN terminology, of a business process. We will only consider the *start event* and the *end event*, which starts and ends the process enactment, respectively. Gateways model the branching and merging of activities. There are several types of gateways in BPMN, each of which can be a *branch* gateway if it has a single incoming flow and multiple outgoing flows, or a *merge* gateway if it has multiple incoming flows and a single outgoing flow. We will consider the following gateways: (i) the *parallel branch* gateway that activates all the outgoing flows at the same time instant, (ii) the *parallel merge* gateway that activates the outgoing flow when all the incoming flows have been activated (that is, the parallel merge synchronizes the incoming flows) (iii) the *exclusive branch* gateway that (non-deterministically) activates exactly one out of the (possibly many) outgoing flows, and (iv) the *exclusive merge* gateway that activates the single outgoing flow upon activation of one of the (possibly many) incoming flows. The diamonds representing parallel and exclusive gateways are labeled by '+' and '×', respectively.

A sequence flow, denoted by an arrow, links two flow objects and denotes a control flow relation, that is, it states that the control flow can pass from the source to the target flow object. If there is a sequence flow from x to y, then x is a *predecessor* of y and y is a *successor* of x. A *path* in a BPMN model is a sequence of flow objects such that every pair of consecutive objects is connected by a sequence flow.

We assume that BPMN models are *well-formed*, that is, they satisfy the following properties: (1) every process contains a unique start event and a unique end event, (2) every flow object occurs on a path from the start event to the end event, (3) the start event has exactly one successor and no predecessor, (4) the end event has exactly one predecessor and no successor, (5) branch gateways have exactly one predecessor and at least one successor, while merge gateways have at least one predecessor and exactly one successor, (6) tasks have exactly one predecessor and one successor, and (7) on every cyclic path there is at least one occurrence of a task (that is, no cycles through gateways only are allowed).

In Figure 1 we show the BPMN model of a purchase order process, called PO, describing a interaction pattern between an e-commerce vendor and a customer.

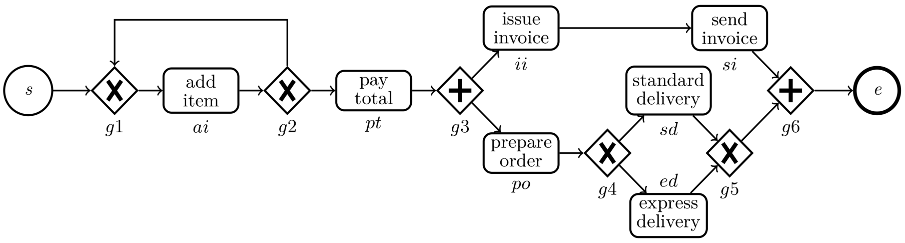

Fig. 1. The BPMN model of the purchase order process *PO*.

At the beginning of the purchase order process the customer adds one or more items to the shopping cart. Then, he pays for all the items, and the vendor (i) issues the invoice and sends it to the customer, and also (ii) prepares the order ships it by a standard or an express delivery method. The process terminates when the invoice has been sent and the order has been delivered.

3 Specification and Semantics of Business Processes

In this section we introduce the notion of a Business Process Specification, which formally represents a business process by means of set of Constrained Horn Clauses, and we define the operational semantics of a BPS.

3.1 Business Process Specification via CHCs

A *Business Process Specification*, or BPS for short, contains: (i) a set of ground facts that specify the flow objects and the sequence flows between them, and (ii) a set of constrained facts that specify the duration of each flow object.

We will use the following predicates: (i) $\mathit{flow\_object}(x)$: x is either a task, or an event, or a gateway; (ii) $\mathit{task}(x)$: x is a task; (iii) $\mathit{start}(x)$ and $\mathit{end}(x)$: x is a start event and an end event, respectively; (iv) $\mathit{exc\_branch}(x)$ and $\mathit{exc\_merge}(x)$: x is an exclusive branch and exclusive merge gateway, respectively; (v) $\mathit{par\_branch}(x)$ and $\mathit{par\_merge}(x)$: x is a parallel branch and a parallel merge gateway, respectively; (vi) $\mathit{seq}(x,y)$: there is a sequence flow from x to y; (vii) $\mathit{duration}(x,d)$: the enactment of the flow object x takes d units of time to be completed.

In the Business Process Specification we assume that: (i) for every task x there exists a single clause of the form $\mathit{duration}(x,d) \leftarrow d_{min} \leq d \leq d_{max}$, where d_{min} and d_{max} are positive integer constants representing the minimal and the maximal time duration of x, respectively, and (ii) for every event and gateway x there exists a single clause of the form $\mathit{duration}(x,0)$ (that is, the enactment of any event or gateway takes no time).

The CHC specification of the BPMN process *PO* of Figure 1 is shown in Table 3.1. Note that a BPS is always satisfiable because it contains no goals, and hence it has a least $\mathbb{Z}$-model.

Our formalization of a BPS also includes a set of clauses that represent the *meta-model* of any BPS. In particular, these meta-model clauses express: (i) the disjointness properties of the sets of its flow objects (for instance, we have the clause: $\mathit{false} \leftarrow \mathit{task}(X), \mathit{par\_branch}(X)$), and (ii) the *well-formedness* properties

```
task(ai).  task(pt).  task(ii).  task(si).  task(po).  task(sd).  task(ed).
start(s).  end(e).  exc_merge(g1).  exc_branch(g2).  par_branch(g3).
exc_branch(g4).  exc_merge(g5).  par_merge(g6).
seq(s,g1).  seq(g1,ai). seq(ai,g2). seq(g2,g1). seq(g2,pt). seq(pt,g3).
seq(g3,ii). seq(g3,po). seq(ii,si). seq(si,g6). seq(po,g4). seq(g4,sd).
seq(sd,g5). seq(g4,ed). seq(ed,g5). seq(g5,g6). seq(g6,e).
duration(s,0).    duration(e,0).    duration(g1,0).   duration(g2,0).
duration(g3,0).   duration(g4,0).   duration(g5,0).   duration(g6,0).
duration(ai,D):- D>=1, D=<6.   % add item
duration(pt,D):- D>=1, D=<2.   % pay total
duration(ii,D):- D>=1, D=<2.   % issue invoice
duration(si,D):- D>=1, D=<3.   % send invoice
duration(po,D):- D>=3, D=<5.   % prepare order
duration(sd,D):- D>=2, D=<4.   % standard delivery
duration(ed,D):- D>=1, D=<3.   % express delivery
```

Table 3.1. BPS for the purchase order process *PO* of Figure 1.

corresponding to Conditions (1)–(7) of Section 2.2. This second set of clauses is as follows:

(c1) $\mathit{eq}(X,Y) \leftarrow \mathit{start}(X), \mathit{start}(Y)$ and $\mathit{eq}(X,Y) \leftarrow \mathit{end}(X), \mathit{end}(Y)$;

(c2) $\mathit{seqq}(S,X) \leftarrow \mathit{start}(S), \mathit{flow\_object}(X)$ and $\mathit{seqq}(X,E) \leftarrow \mathit{flow\_object}(X), \mathit{end}(E)$
where *seqq* is the reflexive, transitive closure of *seq*;

(c3) $\mathit{eq}(Y,Z) \leftarrow \mathit{start}(S), \mathit{seq}(S,Y), \mathit{seq}(S,Z)$ and $\mathit{false} \leftarrow \mathit{start}(S), \mathit{seq}(Y,S)$;

(c4) $\mathit{eq}(Y,Z) \leftarrow \mathit{end}(E), \mathit{seq}(Y,E), \mathit{seq}(Z,E)$ and $\mathit{false} \leftarrow \mathit{end}(E), \mathit{seq}(E,Y)$;

(c5) $\mathit{eq}(Y,Z) \leftarrow \mathit{par\_branch}(X), \mathit{seq}(Y,X), \mathit{seq}(Z,X)$ and
$\mathit{eq}(Y,Z) \leftarrow \mathit{par\_merge}(X), \mathit{seq}(X,Y), \mathit{seq}(X,Z)$
and, similarly, for the *exc_branch* and *exc_merge* gateways;

(c6) $\mathit{eq}(Y,Z) \leftarrow \mathit{task}(X), \mathit{seq}(X,Y), \mathit{seq}(X,Z)$ and
$\mathit{eq}(Y,Z) \leftarrow \mathit{task}(X), \mathit{seq}(Y,X), \mathit{seq}(Z,X)$;

(c7) $\mathit{false} \leftarrow \mathit{gateway\_path}(X,X)$
where $\mathit{gateway\_path}(X,Y)$ is a predicate that holds iff there is a path from X to Y made out of gateways only.

Note that the existence of at least one predecessor and at least one successor for any task or gateway (required by Conditions (5) and (6) of Section 2.2) is enforced by the clauses at Point (c2).

A BPS $\mathcal{B}$ is *well-formed* if clauses (c1)–(c7) hold in the least $\mathbb{Z}$-model of $\mathcal{B}$.

3.2 Operational Semantics

We start off by introducing the notion of a *state* at a time instant t. A state s is a pair $\langle F, t\rangle$, where F is a set of terms, called *fluents*, representing the properties that hold at the time instant t in $\mathbb{Z}$. Let *States* be the set of states.

A fluent is a term of one of the following forms, for any flow object x: (i) $\mathit{begins}(x)$, which represents the beginning of the execution, or enactment, of x, (ii) $\mathit{completes}(x)$, which represents that x has completed its execution, and (iii) $\mathit{enables}(x,y)$, which represents that x upon completion of its execution enables the execution of its successor y, and (iv) $\mathit{enacting}(x,r)$, which represents that the enactment of x requires r units of time to completion (for this reason r is also called the *residual time* of x). From these definitions it follows that $\mathit{begins}(x)$ is equivalent to $\mathit{enacting}(x, r)$, where r is the duration of x, and $\mathit{completes}(x)$ is equivalent to $\mathit{enacting}(x, 0)$. (This redundant representation of fluents allows us to write simpler rules for the operational semantics below.)

The operational semantics is defined by a binary transition relation $\longrightarrow$ which is a subset of $\mathit{States} \times \mathit{States}$ and is derived according to the rules below. In the rules for $\longrightarrow$, besides the predicates introduced in Section 3.1, we use the following ones: (i) $\mathit{not\_par\_branch}(x)$, which holds if x is not a parallel branch, and (ii) $\mathit{not\_par\_merge}(x)$, which holds if x is not a parallel merge.

$$(S_1)\quad \frac{\mathit{begins}(x) \in F \qquad\qquad \mathit{duration}(x, d)}{\langle F, t\rangle \longrightarrow \langle (F \setminus \{\mathit{begins}(x)\}) \cup \{\mathit{enacting}(x, d)\},\ t\rangle}$$

$$(S_2)\quad \frac{\mathit{completes}(x) \in F \qquad\qquad \mathit{par\_branch}(x)}{\langle F, t\rangle \longrightarrow \langle (F \setminus \{\mathit{completes}(x)\}) \cup \{\mathit{enables}(x, s) \mid \mathit{seq}(x, s)\},\ t\rangle}$$

$$(S_3)\quad \frac{\mathit{completes}(x) \in F \qquad \mathit{not\_par\_branch}(x) \qquad \mathit{seq}(x, s)}{\langle F, t\rangle \longrightarrow \langle (F \setminus \{\mathit{completes}(x)\}) \cup \{\mathit{enables}(x, s)\},\ t\rangle}$$

$$(S_4)\quad \frac{\forall p\ \mathit{seq}(p, x) \rightarrow \mathit{enables}(p, x) \in F \qquad\qquad \mathit{par\_merge}(x)}{\langle F, t\rangle \longrightarrow \langle (F \setminus \{\mathit{enables}(p, x) \mid \mathit{enables}(p, x) \in F\}) \cup \{\mathit{begins}(x)\},\ t\rangle}$$

$$(S_5)\quad \frac{\mathit{enables}(p, x) \in F \qquad \mathit{not\_par\_merge}(x)}{\langle F, t\rangle \longrightarrow \langle (F \setminus \{\mathit{enables}(p, x)\}) \cup \{\mathit{begins}(x)\},\ t\rangle}$$

$$(S_6)\quad \frac{\mathit{enacting}(x, 0) \in F}{\langle F, t\rangle \longrightarrow \langle (F \setminus \{\mathit{enacting}(x, 0)\}) \cup \{\mathit{completes}(x)\},\ t\rangle}$$

$$(S_7)\quad \frac{\mathit{no\_other\_premises}(F) \qquad \exists x\, \exists r\ \mathit{enacting}(x, r) \in F \qquad m > 0}{\langle F, t\rangle \longrightarrow \langle F \ominus m,\ t+m\rangle}$$

where: (i) $\mathit{no\_other\_premises}(F)$ holds iff none of the rules S_1–S_6 has its premise true, (ii) $m = \mathit{min}\{r \mid \mathit{enacting}(x, r) \in F\}$, and (iii) $F \ominus m$ is the set F of fluents where every $\mathit{enacting}(x, r)$ is replaced by $\mathit{enacting}(x, r-m)$.

Note that rule (S_7) is the only rule that formalizes the flow of time, as it infers transitions of the form $\langle F, t\rangle \longrightarrow \langle F', t+m\rangle$, with $m > 0$. In contrast, rules (S_1)–(S_6) infer instantaneous state transitions of the form $\langle F, t\rangle \longrightarrow \langle F', t\rangle$.

Now let us explain the meaning of rules (S_1)–(S_7).

(S_1) If the execution of a flow object x begins at time t, then, at the same time t, x is enacting and its residual time is the duration d of x;

(S_2) If the execution of the parallel branch x completes at time t, then x enables *all its successors* at time t;

(S_3) If the execution of x completes at time t and x is not a parallel branch, then x enables *precisely one of its successors* at time t (in particular, this case occurs when x is a task);

(S_4) If *all* the predecessors of x have enabled the parallel merge x at time t, then the execution of x begins at time t;

(S_5) If *at least one* predecessor p of x enables x at time t and x is not a parallel merge, then the execution of x begins at time t (in particular, this case occurs when x is a task);

(S_6) If a flow object x is enacting at time t with residual time 0, then the execution of x completes at time t;

(S_7) Let us assume that at time t: (i) none of rules (S_1)–(S_6) can be applied, (ii) there at least one task whose execution requires r (>0) units of time to get to completion (recall that among the flow objects, tasks only may have positive residual time), and (iii) m is the least among the residual times of all the tasks which are in execution (that is, enacting). Then every task x that is in execution at time t with residual time r, is in execution at time $t+m$ with residual time $r-m$.

We say that state $\langle F', t'\rangle$ is *reachable* from state $\langle F, t\rangle$, if $\langle F, t\rangle \longrightarrow^* \langle F', t'\rangle$, where $\longrightarrow^*$ denotes the reflexive, transitive closure of the transition relation $\longrightarrow$. The *initial state* is the pair $\langle\{\mathit{begins}(s)\}, 0\rangle$, where s denotes the start event.

Note that in our formalization we cannot represent multiple, concurrent executions of the same flow object, because a state is a *set* of fluents. However, this limitation can easily be overcome by considering *multisets* of fluents.

4 Encoding Time-Dependent Properties into CHCs

In this section we show the CHC *interpreter* that encodes the operational semantics of business processes and we show how to encode the time-dependent properties to be verified. We also briefly present two transformation techniques:

(RI): a technique for performing the *removal of the interpreter* (see [11,28] for more details), whereby deriving a set of clauses that can be submitted to automatic tools for satisfiability checking such as the Z3 [12] or the Eldarica [18] solvers for CHCs, and

(PE): a technique for reducing the size of the set of CHCs generated by the RI technique. This PE technique is based on a suitable notion of *predicate equivalence* (see Section 4.3) that may be used, if so desired, for improving the time and space efficiency of the satisfiability checking.

4.1 Encoding the Operational Semantics in CHCs

A state $\langle F, t\rangle$ of the operational semantics is encoded by a term of the form `s(F,T)`, where `F` is a list encoding the set F of fluents and `T` encodes the time instant t at which the fluents of F hold. The transition relation $\longrightarrow$ between states and its reflexive, transitive closure $\longrightarrow^*$ are encoded by the binary predicates `tr` and `reach`, respectively, whose defining clauses are shown in Table 4.2. In the body of the clauses, we have underlined the atoms that encode the premises of the rules of the operational semantics.

The predicate `member(X,L)` selects an element `X` from the list `L`. The predicate `update(F,R,A,FU)` holds iff `FU` is the list obtained from the list `F` by removing all the elements of `R` and adding all the elements of `A`. The predicate `no_other_premises(F)` holds iff the premise of every rule in $\{$`S1`$,\dots,$`S6`$\}$

is false. The predicate `mintime(Enacts,M)` holds iff `Enacts` is a list of terms of the form `enacting(X,R)` and `M` is the minimum value of `R` for the elements of `Enacts`. The predicate `decrease_residual_times(Enacts,M,EnactsU)` holds iff `EnactsU` is the list of terms obtained by replacing every element of `Enacts`, of the form `enacting(X,R)`, by the term `enacting(X,RU)` where `RU=R-M`. The predicates `sublist(S,L)` and `findall(X,G,L)` have the usual meaning.

```
S1. tr(s(F,T), s(FU,T)) :- member(begins(X),F), duration(X,D),
                           update(F,[begins(X)],[enacting(X,D)],FU).
S2. tr(s(F,T), s(FU,T)) :- member(completes(X),F), par_branch(X),
                           findall(enables(X,S),(seq(X,S)),Enbls),
                           update(F,[completes(X)],Enbls,FU).
S3. tr(s(F,T), s(FU,T)) :- member(completes(X),F), not_par_branch(X),seq(X,S),
                           update(F,[completes(X)],[enables(X,S)],FU).
S4. tr(s(F,T), s(FU,T)) :- member(enables(_,X),F), par_merge(X),
                           findall(enables(P,X),(seq(P,X)),Enbls),
                           sublist(Enbls,F), update(F,Enbls,[begins(X)],FU).
S5. tr(s(F,T), s(FU,T)) :- member(enables(P,X),F), not_par_merge(X),
                           update(F,[enables(P,X)],[begins(X)],FU).
S6. tr(s(F,T), s(FU,T)) :- member(enacting(X,R),F), R=0,
                           update(F,[enacting(X,R)],[completes(X)],FU).
S7. tr(s(F,T), s(FU,TU)) :- no_other_premises(F), member(enacting(_,_),F),
                           findall(Y,(Y=enacting(X,R),member(Y,F)),Enacts),
                           mintime(Enacts,M), M>0,
                           decrease_residual_times(Enacts,M,EnactsU),
                           update(F,Enacts,EnactsU,FU), TU=T+M.
R1. reach(S,S).
R2. reach(S,S2) :- tr(S,S1), reach(S1,S2).
```

Table 4.2. The CHC interpreter for the operational semantics of time-aware BPs.

4.2 Encoding Time-Dependent Properties

By using the `reach` predicate and integer constraints, we can specify many useful time-dependent properties. In particular, we can specify safety properties (stating that 'no unsafe state can be reached'), schedulability properties (stating that a process will be completed within a given deadline), response properties (stating that, whenever a task is executed, another task will be executed within a given time).

In order to see how we encode time-dependent properties of business processes, we consider a property of the process PO stating that, whenever the customer pays and the process PO completes, then completion occurs within 9 time units after payment. By using the reachability relation $\longrightarrow^*$, this property can be written as follows:

Q: if $\langle\{\mathit{begins}\,(s)\},0\rangle \longrightarrow^* \langle\{\mathit{completes}(pt)\},t_{pt}\rangle \longrightarrow^* \langle\{\mathit{completes}(e)\},t_e\rangle$,
then $t_e \leq t_{pt}+9$

The reader can check that Q holds for the process PO because, in the worst case, the time needed for preparing and delivering the order is actually 9 time units and this time is greater than the time needed for issuing and sending the invoice, which is 5 time units. The property Q is encoded by the following goal (where `s(_,_)` is the constructor for states, while the constant `s` of arity 0 denotes the start event):

```
Q.  false :- Ts=0, Tpt>Ts, Te>Tpt+9,
             reach(s([begins(s)],Ts), s([completes(pt)],Tpt)),
             reach(s([completes(pt)],Tpt), s([completes(e)],Te)).
```

The clauses `S1`–`S7`,`R1`,`R2`,`Q`, together with the clauses encoding the process PO, will be collectively referred to as the *interpreter* I. We have that the property Q is valid for the process PO iff the set I of CHCs is satisfiable.

Despite several tools have been developed for checking the satisfiability of constrained Horn clauses, none of them can effectively be leveraged in our example. Constraint logic programming systems [20] are focused on proving the unsatisfiability of sets of clauses, rather then their satisfiability, and they may fail to terminate for the given set I because a clause for `reach` is recursive (note, in particular that the `add_item` task can be executed an unbounded number of times). State-of-the-art CHC solvers [12,18] also fail because the predicates in I are defined over lists and structured terms (not just integers) and they depend on the `findall` predicate, which is not available in those solvers.

In order to be able to effectively use off-the-shelf CHC solvers for checking the validity of time-dependent properties, we apply the so-called *removal of the interpreter* transformation, denoted RI [11,28]. This transformation is a program specialization strategy based on unfold/fold transformation rules, which takes the program I as input and produces as output a program I_{sp} that is equivalent to I with respect to satisfiability. Indeed, by the correctness of the unfold/fold transformation rules [13], we have that I is satisfiable iff I_{sp} is satisfiable.

A notable effect of applying the transformation RI, which removes the interpreter, is that the program I_{sp} contains no occurrences of the predicates and terms used for encoding the operational semantics and the process PO. In particular, the clauses of I_{sp} will be of the form $A \leftarrow c, G$, where the arguments of the atoms are variables and c is a constraint. For instance, the goal `Q` expressing the property Q above is transformed into the following goal:

```
Q1.  false :- A=0, B=<2, C=<6, D=<5, E>0, F-E>9, B>=1, C>=1, D>=3,
             new1(C,A,E), new2(B,D,E,F).
```

The new predicates `new1` and `new2` have been introduced by the *definition rule*, and the extra constraints have been derived by the *unfolding rule*. We refer to [11] for the details of the transformation. The whole set of clauses derived by the transformation RI is listed in the online Appendix A.1[4] . The satisfiability of this derived set of clauses can be proved in a fully automatic way by using the Z3 CHC solver, as it will be shown in Section 5.

4.3 Predicate Equivalence

Now we present a transformation, called *predicate equivalence*, denoted PE, that allows us to reduce the size of a set of constrained Horn clauses when suitable

[4] Available at `http://map.uniroma2.it/lopstr16/appendix.pdf`

equivalences between predicates hold. Since predicate equivalence is undecidable in general, we introduce a restricted, decidable notion of equivalence based on constraint equivalence and predicate renaming.

First we need some preliminary notions. Let $\exists Y\,(c_1, G_1)$ and $\exists Z\,(c_2, G_2)$ be two existentially quantified conjunctions of constraints and atoms, where $Y \cap \mathit{vars}(c_2, G_2) = \emptyset$ and $Z \cap \mathit{vars}(c_1, G_1) = \emptyset$ (we extend, in the obvious way, to tuples of variables notions defined for variables and sets of variables). We say that $\exists Y\,(c_1, G_1)$ and $\exists Z\,(c_2, G_2)$ are *equivalent modulo constraints*, if there exists a renaming substitution $\{Y'/Z'\}$ for (c_1, G_1), with $Y' \subseteq Y$ and $Z' \subseteq Z$, such that:
(i) $G_1\{Y'/Z'\} = G_2$, modulo reordering of atoms, and
(ii) $\mathbb{Z} \models \forall(\exists U\, c_1\{Y'/Z'\} \leftrightarrow \exists V\, c_2)$, where $U = Y - Y'$ and $V = Z - Z'$.

For instance, $\exists Y\,(X \geq Y,\ p(X, Y))$ and $\exists V, W\,(X \geq V,\ V \geq W,\ p(X, W))$ are equivalent modulo constraints. Clearly, if $\exists Y\,(c_1, G_1)$ and $\exists Z\,(c_2, G_2)$ are equivalent modulo constraints, then $\mathbb{Z} \models \forall(\exists Y\,(c_1, G_1) \leftrightarrow \exists Z\,(c_2, G_2))$.

Let P be a set of CHCs. By $\mathit{Pred}(P)$ we denote the set of predicate symbols occurring in P. A *predicate renaming* for P is a, possibly not injective, mapping $\pi\colon \mathit{Pred}(P) \rightarrow Q$, where Q is a set of predicate symbols. Given a set S of formulas with predicates in $\mathit{Pred}(P)$, $\pi(S)$ is a new set of formulas obtained by replacing, for all predicates $p \in \mathit{Pred}(P)$, every occurrence of p in S by $\pi(p)$.

For every $k \geq 1$, let X be a fixed k-tuple of distinct variables. Without loss of generality, we assume that for every k-ary predicate $p \in \mathit{Pred}(P)$, all clauses are of the form $p(X) \leftarrow B$, where B is a conjunction of constraints and atoms. By $\mathit{Bodies}(p(X), P)$ we denote the set $\{B \mid p(X) \leftarrow B \text{ is a clause in } P\}$. We write $\mathit{Bodies}(p(X), P) \equiv \mathit{Bodies}(q(X), P)$ if there exists a bijection $\eta\colon \mathit{Bodies}(p(X), P) \rightarrow \mathit{Bodies}(q(X), P)$ such that, for every $B \in \mathit{Bodies}(p(X), P)$, $\exists Y\, B$ and $\exists Z\, \eta(B)$ are equivalent modulo constraints, where Y is the tuple of variables occurring in B and not in X, and Z is the tuple of variables occurring in $\eta(B)$ and not in X.

Definition 1 (Predicate Equivalence). *Let P be a set of clauses and $E = \{P_1, \ldots, P_n\}$ be a partition of $\mathit{Pred}(P)$. For $i = 1, \ldots, n$, let e_i be a predicate symbol in P_i, and $\pi : \mathit{Pred}(P) \rightarrow \{e_1, \ldots, e_n\}$ be a predicate renaming for P such that, for $i = 1, \ldots, n$, $\pi(p) = e_i$ iff $p \in P_i$.*

The partition E is a cp-equivalence *on P if, for $i = 1, ..., n$, given any two predicates p, q in P_i, p and q have the same arity k and, for any fixed k-tuple X of distinct variables, $\pi(\mathit{Bodies}(p(X), P)) \equiv \pi(\mathit{Bodies}(q(X), P))$.*

Note that one can compute the coarsest cp-equivalence on P by a greatest fixpoint construction starting from the partition where all predicate symbols belong to the same equivalence class.

Given a cp-equivalence E on P together with the predicate renaming π considered in Definition 1, we can transform P into a set $\widetilde{\pi}(P, E)$ of clauses in two steps: (i) we remove from P all clauses whose head predicate does not appear in the range of π, and (ii) we apply π to the remaining clauses.

Theorem 1. *For any cp-equivalence E on a set P of clauses, P is satisfiable iff $\widetilde{\pi}(P, E)$ is satisfiable.*

Checking the satisfiability of $\widetilde{\pi}(P, E)$ is often more efficient than checking the satisfiability of P, specially when we use solvers, like Z3, that construct a model of each predicate. Indeed, when checking the satisfiability of $\widetilde{\pi}(P, E)$, the solver has to construct, for each equivalence class E, a model of one predicate only.

To see an example of cp-equivalence, let us consider the following subset of the 51 clauses derived by the removal of the interpreter in our *PO* example (the complete listing of those clauses is given in the online Appendix A.2[5]):

```
new5(A,B,C,D) :- A=0, new21(B,C,D).
new5(A,B,C,D) :- A=0, B=0, E=<3, E>=1, new10(E,C,D).
new5(A,B,C,D) :- B=0, E=<3, E>=1, new7(A,E,C,D).
new5(A,B,C,D) :- E=0, F=-A+B, G=A+C, A-B=<0, A>0, new5(E,F,G,D).
new5(A,B,C,D) :- E=0, F=A-B, G=B+C, B>0, A-B>=0, new5(F,E,G,D).
new4(A,B,C,D) :- A=0, new21(B,C,D).
new4(A,B,C,D) :- A=0, B=0, E=<3, E>=1, new10(E,C,D).
new4(A,B,C,D) :- B=0, E=<3, E>=1, new6(A,E,C,D).
new4(A,B,C,D) :- E=0, F=-A+B, G=A+C, A-B=<0, A>0, new4(E,F,G,D).
new4(A,B,C,D) :- E=0, F=A-B, G=B+C, B>0, A-B>=0, new4(F,E,G,D).
```

The partition $E = \{\{\texttt{new5}, \texttt{new4}\}, \{\texttt{new7}, \texttt{new6}\}, \{\texttt{new21}\}, \{\texttt{new10}\}\}$ of the set of predicates occurring in the above clauses is a cp-equivalence. The predicate renaming associated with E is:

$\pi(\texttt{new5}) = \pi(\texttt{new4}) = \texttt{new4}$ $\quad$ $\pi(\texttt{new7}) = \pi(\texttt{new6}) = \texttt{new6}$
$\pi(\texttt{new21}) = \texttt{new21}$ $\quad$ $\pi(\texttt{new10}) = \texttt{new10}$.

By applying the predicate equivalence transformation to the whole set of 51 clauses, we get an equisatisfiable set of 35 clauses. In particular, in the resulting set the clauses for `new5` are no longer present and all occurrences of `new5` are replaced by `new4`.

5 Automated Verification

We have implemented the *Removal of the Interpreter* (RI) and the *Predicate Equivalence* (PE) transformations presented in Section 4.2 and Section 4.3, respectively, by using the VeriMAP transformation system [10].

We use these transformations for verifying properties of business processes in the following two different ways:

(i) 'RI; Z3', that is, we execute RI, and then we check the satisfiability of the clauses generated by RI by applying the solver Z3[6] [12], and
(ii) 'RI; PE; Z3', that is, we execute RI, then PE, and finally we check the satisfiability of the clauses generated by PE by applying the solver Z3.

In Table 5.3 we report the results obtained by using our prototype implementation for the following business processes:

(1) the Purchase Order (*PO*) shown in Figure 1, consisting of 7 tasks, 6 gateways, and 17 flows,
(2) the Request Day Off Approval (*RDOA*), adapted from [19], consisting of 7 tasks, 4 gateway, 14 flows and representing the activities involving a company's leadership to approve an employee's request for a day off,

[5] Available at `http://map.uniroma2.it/lopstr16/appendix.pdf`

[6] v4.4.2, master branch as of 2016-02-18, with the Duality fixed-point engine [25]

(3) the ST-segment Elevation Myocardial Infarction (*STEMI*), adapted from [7], consisting of 11 tasks, 6 gateways, 22 flows and representing an excerpt of the triage process for hospital admission, and

(4) the *STEMI* with Coronary Care Unit admission (*STEMI*+*CCU*), adapted from [8], consisting of 26 tasks, 18 gateways, and 52 flows and representing an extension of *STEMI* which also includes the activities for admitting a patient to the Coronary Care Unit.

For these processes we have considered ten temporal properties (denoted $P1$–$P10$ in Table 5.3)[7], each one being of the form: *if* some reachability properties between states hold, *then* some constraints between their associated time instants hold.

The experiments have been performed on an Intel Core i5-2467M 1.60GHz processor with 4GB of memory under GNU/Linux OS.

Business Process	Property	RI		Z3 time1	answer	PE		Z3 time2	cls reduction	time speedup
		time	cls			time	cls			
PO	$P1$	0.49	51	0.82	true	0.05	35	0.57	0.31	1.44
	$P2$	0.27	51	0.68	true	0.05	37	0.53	0.27	1.28
	$P3$	0.35	12	0.10	false	0.04	12	0.10	0.00	1.00
RDOA	$P4$	0.14	20	0.31	false	0.03	16	0.22	0.20	1.41
STEMI	$P5$	0.34	52	1.04	true	0.05	43	0.88	0.17	1.18
	$P6$	0.31	7	0.09	false	0.02	7	0.09	0.00	1.00
	$P7$	0.36	67	1.62	true	0.06	56	1.60	0.16	1.01
STEMI +*CCU*	$P8$	1.58	226	10.70	true	0.17	181	9.75	0.20	1.10
	$P9$	0.14	29	30.17	false	0.03	23	11.62	0.21	2.60
	$P10$	0.10	15	2.08	false	0.03	15	2.08	0.00	1.00

Table 5.3. Columns 'RI.time' and 'RI.cls' denote the time taken by RI and the number of clauses generated by RI, respectively. Column 'Z3.time1' denotes the time taken by Z3 when executed after RI. Column 'answer' tells us whether or not the property holds. Columns 'PE.time' and 'PE.cls' denote the time taken by PE and the number of clauses generated by PE, respectively. Column 'Z3.time2' denotes the time taken by Z3 when executed after PE. The reduction of the number of clauses (cls reduction) is $\frac{\text{RI.cls} - \text{PE.cls}}{\text{RI.cls}}$ and the time speedup is $\frac{\text{Z3.time1}}{\text{Z3.time2}}$. Times are in seconds.

In Table 5.3 we have *not* reported the results of applying the solver Z3 directly to the clauses encoding the given business processes and properties. Indeed, as already mentioned in Section 4.2, Z3 is not able to prove the satisfiability of those clauses, if one does not first apply the transformation RI.

The transformation RI is quite efficient and takes less than half a second for all properties with the exception of property $P8$, which generates 226 clauses. The time taken by Z3 for the verification of the properties (with or without the preliminary application of PE) is generally small (indeed, it is not greater than 1.62 seconds), with the exception of properties $P8$–$P10$ referring to the most complex business process we have considered, which is the *STEMI*+*CCU* process.

[7] The VeriMAP tool and the encodings of the examples of Table 5.3 are available at `http://map.uniroma2.it/lopstr16/VeriMAP_lopstr16-linux_x86_64.tar.gz`

Note also that the transformation PE often reduces the number of clauses generated by RI and speeds up the satisfiability check performed by Z3. Moreover, in our examples PE never deteriorates the total verification time in any significant way, in the sense that the time taken by 'RI; PE; Z3' is never significantly greater than the time taken by 'RI; Z3'.

6 Related Work

Several papers have proposed approaches to model business processes with time constraints and, in particular, duration [1,7,15,16,35] (see [6] for a recent survey).

The approach of Arbab et al. [1] provides a translation of BPMN into the coordination language Reo. Due to Constraint Automata semantics of Reo, in principle this translation allows formal reasoning about BPMN processes depending on time and resources. However, the paper does not provide any formalized verification technique.

The workflow conceptual model proposed by Combi and Posenato [7] enables the specification and analysis of time constraints in business processes. They propose temporal constructs for expressing various kinds of time constraints, and also introduce the notion of controllability for workflow schemata. Controllability ensures the executability of a workflow for any duration of the tasks performed by the 'external world'. Unfortunately, the algorithms for testing controllability presented in [7] may require a costly, exhaustive exploration of the search space.

Gonzalez del Foyo and Silva consider workflow diagrams extended with task durations and the latest execution deadline of each task [15]. They provide a translation into Time Petri Nets [3] (where clocks are associated with transitions in the net) and use the tool TINA [4] to answer schedulability questions.

The approach proposed by Gagné and Trudel [16] enables the specification of temporal constraints (such as 'As Soon As Possible') and temporal dependencies. However, unlike the approach presented here, no automated verification mechanism of time-dependent properties is provided.

The approach proposed by Wong and Gibbons [35] uses a timed semantic function which takes a diagram describing a collaboration, and returns a CSP process [17] that models the timed behavior of that diagram, by using the notion of a relative time in the form of delays chosen non-deterministically within given intervals. Properties are then verified by using the FDR system [14].

The proposal by Watahiki et al. [32] and other proposals surveyed in [6] use Timed Automata to model business processes with time constraints. They also use the UPPAAL tool [22] for the automatic proof of the properties of interest.

As already mentioned in the Introduction, the translations into formalisms such as Timed Automata, Time Petri Nets, and CSP, may not be adequate when taking into consideration properties of business processes that require general purpose logical reasoning.

Finally, we would like to mention work on modeling and analyzing business processes with explicit time representation based on the *Event Calculus* [21] (see, for instance, [26]). However, the Event Calculus lacks a simple translation into constrained Horn clauses (in particular, it makes use of negation), and hence it cannot be directly handled by CHC solvers.

7 Conclusions

We have presented a logic-based language to specify BPMN models where time and duration of activities are explicitly represented. The language enables the specification of time constraints, given in the form of lower and upper bounds associated with the duration of tasks. These are useful features with an intuitive meaning that allow the specifier to annotate activities with some time restrictions. The language supports the specification of a wide range of time-dependent properties such as the schedulability and the response time.

The main advantage of our approach is that it allows us to automatically generate constrained Horn clauses from the formal definition of the semantics of the BPMN models and the time-dependent properties of interest. Then, by exploiting recent advances in the field of CHC solving, we get very effective reasoning tools for verifying properties of business processes. Finally, since our approach is parametric with respect to the language used for modeling processes, it is possible to incorporate various extensions of that language with little effort.

References

1. F. Arbab, N. Kokash, and S. Meng. Towards using Reo for compliance-aware business process modeling. In Proc. *ISoLA '08*, CCIS 17, pp. 108–123. Springer, 2008.
2. B. Bagheri Hariri, D. Calvanese, G. De Giacomo, A. Deutsch, and M. Montali. Verification of relational data-centric dynamic systems with external services. In Proc. *PODS '13*, pages 163–174, 2013.
3. B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.
4. B. Berthomieu and F. Vernadat. Time Petri nets analysis with TINA. In Proc. *QEST '06*, pages 123–124. IEEE Computer Society, 2006.
5. N. Bjørner, A. Gurfinkel, K. L. McMillan, and A. Rybalchenko. Horn clause solvers for program verification. In *Fields of Logic and Computation II - Essays Dedicated to Yuri Gurevich*, LNCS 9300, pages 24–51. Springer, 2015.
6. S. Cheikhrouhou, S. Kallel, N. Guermouche, and M. Jmaiel. The temporal perspective in business process modeling: a survey and research challenges. *Service Oriented Computing and Applications*, 9(1):75–85, 2015.
7. C. Combi and R. Posenato. Controllability in Temporal Conceptual Workflow Schemata. In Proc. *BPM '09*, LNCS 5701, pages 64–79. Springer, 2009.
8. C. Combi, M. Gozzi, R. Posenato, and G. Pozzi. Conceptual Modeling of Flexible Temporal Workflows. *ACM Trans. Auton. Adapt. Syst.*, 7(2):19:1–19:29, July 2012.
9. E. Damaggio, A. Deutsch, and V. Vianu. Artifact systems with data dependencies and arithmetic. *ACM Trans. Database Syst.*, 37(3):1–36, 2012.
10. E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. VeriMAP: A Tool for Verifying Programs through Transformations. In Proc. *TACAS '14*, LNCS 8413, pages 568–574. Springer, 2014. `www.map.uniroma2.it/VeriMAP`.
11. E. De Angelis, F. Fioravanti, A. Pettorossi, and M. Proietti. Semantics-based generation of verification conditions by program specialization. Science of Computer Programming (to appear). Elsevier, 2017.
12. L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In Proc. *TACAS '08*, LNCS 4963, pages 337–340. Springer, 2008.
13. S. Etalle and M. Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166:101–146, 1996.

14. Formal Systems (Europe) Ltd. Failures-Divergences Refinement, FDR2 User Manual. www.fsel.com, 1998.
15. P. M. Gonzalez del Foyo and J. R. Silva. Using time Petri nets for modelling and verification of timed constrained workflow systems. In Proc. *ABCM Symposium Series in Mechatronics*, pages 471–478. Springer, 2008.
16. D. Gagné and A. Trudel. Time-BPMN. In Proc. *CEC '09*, pages 361–367. IEEE Computer Society, 2009.
17. C. A. R. Hoare, Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, ACM, New York, USA, 1978.
18. H. Hojjat, F. Konecný, F. Garnier, R. Iosif, V. Kuncak, and P. Rümmer. A verification toolkit for numerical transition systems. In Proc. *FM '12*, LNCS 7436, pages 247–251. Springer, 2012.
19. W. Huai, X. Liu, and H. Sun. Towards Trustworthy Composite Service Through Business Process Model Verification. In Proc. *UIC-ATC '10*, pages 422–427. IEEE Computer Society, 2010.
20. J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.
21. R. A. Kowalski and M. J. Sergot. A logic-based calculus of events. *New Generation Comput.*, 4(1):67–95, 1986.
22. K. G. Larsen, P. Pettersson, and Wang Yi. Uppaal in a Nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1(1–2):134–152, October 1997.
23. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987. Second Edition.
24. M. Makni, S. Tata, M.M. Yeddes, and N. Ben Hadj-Alouane. Satisfaction and coherence of deadline constraints in inter-organizational workflows. In Proc. *OTM '10*, LNCS 6426, pages 523–539. Springer, 2010.
25. K. McMillan and A. Rybalchenko. Computing relational fixed points using interpolation. *Technical Report MSR-TR-2013-6*, Microsoft Research, January 2013.
26. M. Montali, F. Maggi, F. Chesani, P. Mello, and Wil M. P. van der Aalst. Monitoring business constraints with the Event Calculus. *ACM Trans. Intell. Syst. Technol.*, 5(1):17:1–17:30, January 2014.
27. OMG. Business Process Model and Notation. www.omg.org/spec/BPMN/, 2013.
28. J. C. Peralta, J. P. Gallagher, and H. Saglam. Analysis of Imperative Programs through Analysis of Constraint Logic Programs. In Proc. *SAS '98*, LNCS 1503, pages 246–261. Springer, 1998.
29. M. Proietti and F. Smith. Reasoning on data-aware business processes with constraint logic. In Proc. *SIMPDA '14*, Vol. 1293 of *CEUR*, pages 60–75, 2014.
30. F. Smith and M. Proietti. Rule-based behavioral reasoning on semantic business processes. In Proc. *ICAART '13, Vol. II*, pages 130–143. SciTePress, 2013.
31. A. M. ter Hofstede, W. M. P. van der Aalst, M. Adams, and N. Russell, eds. *Modern Business Process Automation: YAWL and its Support Environment.* Springer, 2010.
32. K. Watahiki, F. Ishikawa, and K. Hiraishi. Formal verification of business processes with temporal and resource constraints. In Proc. *IEEE Intern. Conf. on Systems, Man and Cybernetics*, pages 1173–1180. IEEE, 2011.
33. I. Weber, J. Hoffmann, and J. Mendling. Beyond soundness: On the verification of semantic business process models. *Distrib. Parallel Databases*, 27:271–343, 2010.
34. M. Weske. *Business Process Management: Concepts, Languages, Architectures.* Springer, 2007.
35. P.Y.H. Wong and J. Gibbons. A relative timed semantics for BPMN. *Electr. Notes Theor. Comput. Sci.*, 229(2):59–75, 2009.