



A framework for automated conflict detection and resolution in medical guidelines [☆]



J. Bowles ^{a,*}, M.B. Caminati ^a, S. Cha ^b, J. Mendoza ^a

^a School of Computer Science, University of St Andrews, Jack Cole Building, St Andrews KY16 9SX, United Kingdom

^b Automation and Information Systems, Technical University of Munich, Germany

ARTICLE INFO

Article history:

Received 15 April 2018

Received in revised form 9 June 2019

Accepted 1 July 2019

Available online 10 July 2019

Keywords:

Clinical guidelines

Formal methods

SMT solvers

Theorem provers

Isabelle/HOL

ABSTRACT

Common chronic conditions are routinely treated following standardised procedures known as *clinical guidelines*. For patients suffering from two or more chronic conditions, known as multimorbidity, several guidelines have to be applied simultaneously, which may lead to severe adverse effects when the combined recommendations and prescribed medications are inconsistent or incomplete. This paper presents an automated formal framework to detect, highlight and resolve conflicts in the treatments used for patients with multimorbidities focusing on medications. The presented extended framework has a front-end which takes guidelines captured in a standard modelling language and returns the visualisation of the detected conflicts as well as suggested alternative treatments. Internally, the guidelines are transformed into formal models capturing the possible unfoldings of the guidelines. The back-end takes the formal models associated with multiple guidelines and checks their correctness with a theorem prover, and inherent inconsistencies with a constraint solver. Key to our approach is the use of an optimising constraint solver which enables us to search for the best solution that resolves/minimises conflicts according to medication efficacy and the degree of severity in case of harmful combinations, also taking into account their temporal overlapping. The approach is illustrated throughout with a real medical example.

© 2020 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (<http://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In healthcare management and practice, as in other domains, clinical and medical procedures are streamlined by adopting standardised guidelines. In particular, treatments for common chronic conditions have been subject to various clinical trials, and the outcomes documented in *clinical guidelines* (CGs) specifying accepted treatment steps, possible alternatives, and recommendations to follow. In the UK, the National Institute of Health and Care Excellence (NICE¹) for England and Wales, and the Scottish Intercollegiate Guidelines Network (SIGN²) for Scotland, publish CGs for most well known conditions. Clinical guidelines play an important role in improving healthcare for people with long-term conditions. However,

[☆] This research is supported by the MRC-funded UK Research and Innovation grant MR/S003819/1 and by EPSRC grant EP/M014290/1.

* Corresponding author.

E-mail addresses: jkfb@st-andrews.ac.uk (J. Bowles), mbc8@st-andrews.ac.uk (M.B. Caminati), suhyun.cha@tum.de (S. Cha), jjm20@st-andrews.ac.uk (J. Mendoza).

¹ NICE www.nice.org.uk.

² SIGN www.sign.ac.uk.

in people with complex needs and two or more chronic conditions simultaneously (aka *multimorbidity*), current guideline recommendations rapidly lead to a need to take multiple medications (known as *polypharmacy* when it means more than 5 medications) without providing guidance on how best to prioritise recommendations [25]. Multimorbidity is also becoming increasingly common. In Scotland, over half of all people with chronic conditions have multimorbidity [20]. When managing the treatment plans for such patients the problem is how to resolve possible contradictions that may arise when combining several guidelines. Conflicts may correspond to the simultaneous administration of drugs that are known to have adverse reactions, or inconsistencies in health recommendations. There is little information on how to handle conflicts when they arise, or what alternatives to suggest instead.

In recent work, we have explored how formal methods can help with the development of an automated framework that combines efficient and formal verification techniques, such as constraint solvers and theorem provers, to identify steps in different CGs that cause problems if carried out together (e.g., two drugs prescribed for different conditions may interact, food may interact with a drug, health recommendations may contradict each other) whilst at the same time find preferred alternatives according to certain criteria (e.g., drug efficacy, prevalent disease, patient allergies, preferences, etc.). Future integration of such techniques in practice can lead to the development of clinical decision support systems to manage treatments for patients with complex needs and multimorbidities. The need for this has been stressed in [25].

Here, we present a framework consisting of a front-end where a guideline, given as a BPMN model, is transformed into our formal model, a Labelled Event Structure (LES). The correctness of the model generated by the transformation is checked with the theorem prover Isabelle/HOL. When two or more BPMN models are fed into the framework it uses SMT solvers technology to identify inconsistencies – which we can highlight – and find ideal alternatives under certain criteria. In this paper the criteria used is known medication effectiveness as well as the severity of harmful combinations. In other words, each medication has an associated positive score, and groups of medications with known adverse reactions have an associated negative score. This score is used by the SMT solver to find ideal solutions with the highest possible score.

This paper extends our work presented in [6] by showing in detail the different components that are involved in the framework to detect inconsistencies between care guidelines for different chronic conditions. In addition, we also include formal, computable Isabelle/HOL definitions for notions related to the underlying event structure model as well as formal Isabelle/HOL theorems establishing the correctness of the generated event structure models. Furthermore, we introduce a novel way of eliciting sub-optimal solutions from the SMT solver Z3 while keeping the SMT-LIB standard language. The results of our underlying framework can be visualised and explored further through a new, javascript-based interactive GUI. An example is used to illustrate the approach throughout.

This paper is structured as follows. In the next section, we describe the overview of the proposed approach. Individual details of the framework are then explored in subsequent sections. Section 3 describes the front-end and how a BPMN model is converted into our underlying formal model namely a labelled event structure. Section 4 shows the back-end concentrating on conflict detection and resolution by finding better alternatives. It includes details on the formal Isabelle/HOL definitions and theorems which are used to ensure the correctness of our approach. We illustrate the approach with an example in Section 5 also used in Section 6 to showcase a technique to elicit an arbitrary number of further solutions besides the optimal one, sorted by the scores. Section 7 discusses related work, and Section 8 concludes the paper.

2. Framework overview

Clinical guidelines, such as those published by NICE, are given in a combination of visual diagrams (flowcharts) and natural language, and capture the steps and decision points taken in the treatment of a disease. For example, if an adult has been diagnosed with type 2 diabetes, then there is one guideline that describes different aspects related to treating diabetes, including the management of blood glucose. The essential steps for the glucose lowering treatment guideline from NICE are shown in Fig. 1, and assume that the patient has already been diagnosed with type 2 diabetes.

NICE guidelines³ are interactive flowcharts, where individual nodes can sometimes be refined further or be clicked on for additional information (typically shown in natural language on the right hand side of the diagram). As an example of the first case, the guideline for Type 2 diabetes in adults overview, contains one node Managing Blood Glucose which when refined displays similar information to what is shown in Fig. 1. In the second case, for the node Initial drug treatment if metformin is contraindicated or not tolerated, which applies to patients who cannot take metformin, three alternative medications are suggested (a DPP-4 inhibitor, pioglitazone or a sulfonylurea).

Metformin is, nonetheless, usually the medication given for an initial drug treatment (node: Initial drug treatment with metformin). When a patient's condition deteriorates, they move to the next step First intensification with metformin combination therapy which adds a further medication to metformin (a DPP-4 inhibitor, pioglitazone or a sulfonylurea). It is important that the medications chosen at any step of the CG are suitable in case the patient has a comorbidity, i.e., another ongoing chronic condition. It is common for patients with type 2 diabetes to have or later develop hypertension, chronic kidney disease and/or cardiovascular disease. An extract of the current recommendation for treating hypertension for persons aged under 55 is shown in Fig. 2. There is an alternative path (not shown) for patients over 55 and of certain ethnic groups where a different group of medications may be used initially instead. Hypertension is one example where disease progression means

³ NICE guidelines (aka pathways) can be found at <https://pathways.nice.org.uk>.

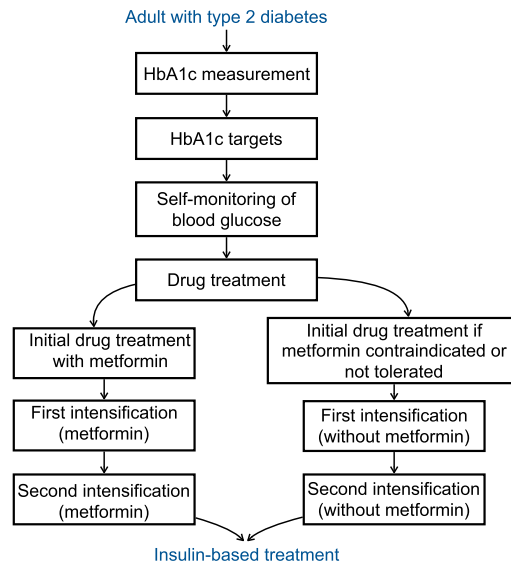


Fig. 1. Managing blood glucose in adults with type 2 diabetes (NICE).

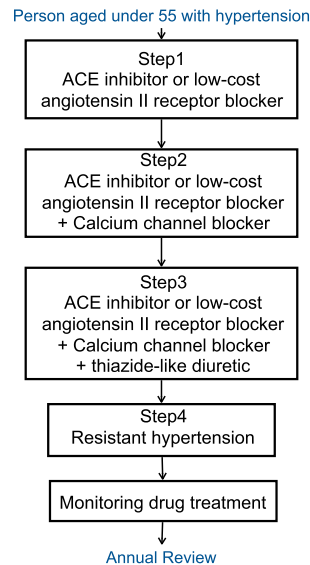


Fig. 2. Treatment steps for hypertension (NICE).

the introduction of further drugs into the treatment which will at some stage become problematic in case of comorbidities. For example, at a certain stage of hypertension, a thiazide-like diuretic may be introduced which has a risk of developing or exacerbating diabetes.

NICE guidelines, similar to what is shown in Fig. 1 for managing blood glucose and Fig. 2 for treating hypertension, are essentially process descriptions showing a sequence, alternatives or repeated steps to follow as applicable. However, the notation used in these, and similar clinical guidelines, has inherent ambiguity. For instance, if a patient in Step 2 of her treatment for hypertension worsens, her prescribed medications should be revised within Step 2 and replaced with available alternatives before evolving to Step 3. This is not evident from looking at the diagram in Fig. 2. Automated solutions require unambiguous notation, and for that reason we use BPMN (Business Process Modeling Notation) [15] to capture the information contained in clinical guidelines. BPMN entails the notation and expressiveness required, and has been given a Petri net based semantics [18] which is in line with our approach (we describe it later in more detail). Note that it is outside the scope of the present paper to validate whether a BPMN model accurately captures clinical practice, but the notation used is sufficient to capture guidelines accurately.

In this paper, we present an integrated formal framework to detect, highlight and resolve conflicts in the treatment of patients with multimorbidity. Our approach combines information from existing CGs for common chronic conditions, such as those provided by NICE or SIGN, and a database of known drug effectiveness and drug interactions (including drug-drug,

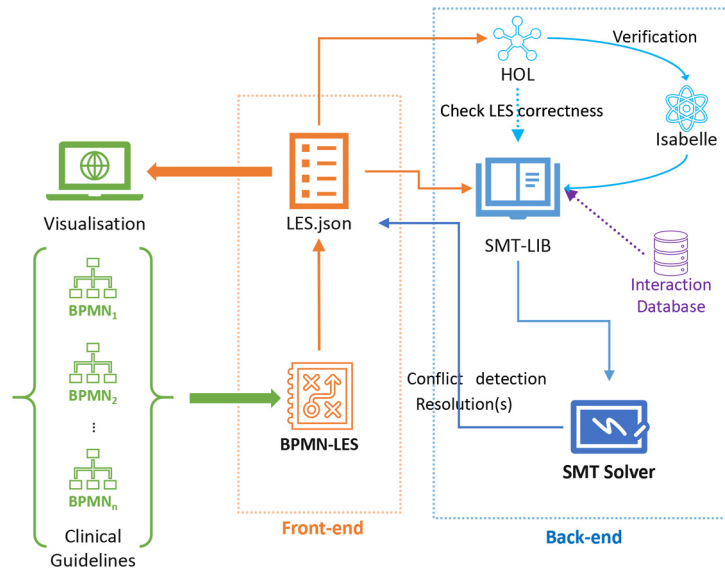


Fig. 3. High-level outline of the framework.

drug-disease) which we give numerical scores. Several sources for drug interactions are publicly available⁴ and we have replicated a database including information on effectiveness and severity as provided by drug companies. This information is used to detect conflicts and also to resolve them.

Our framework is illustrated in Fig. 3, and consists of the following components:

1. A *front-end component* (pointed out in Fig. 3), offering an interface to manipulate the information contained in existing clinical guidelines (CGs) as BPMN models (generating JSON output).
2. A *formal model* to capture the semantics contained in the structure and annotations of the CGs extracted from the front-end stage. This model should be suitable for expressing multiple CGs and formulating the co-existence of conflicts across guidelines.
3. A *back-end component* (pointed out in Fig. 3), implementing the formal model, and capable of applying it to the particular instances of CGs coming from the front-end. This component applies constructions of the formal model to compute and represent the relevant notions of guideline conflicts, as well as compute alternatives that either have no conflicts or have reduced conflicts. Alternative treatment paths are computed with respect to a notion of medication/drug effectiveness score, where drugs have a positive effectiveness score (for a given condition) and a negative score is associated with drug-drug pairs if these drugs have adverse reactions (classified as mild, moderate, severe in the medical domain). The information on drug effectiveness scores and drug interaction severity scores are contained in the interaction database.
4. A *verification component* (see Fig. 3) to formally verify the back-end (e.g., to validate its correctness with respect to specifications, expected behaviour, etc.).

Clinical guidelines are modelled in a domain-specific version of BPMN which we designate $BPMN_{CG}$. Our formal model is based on labelled event structures (LES) [36,27]. LES can be seen as the unfolding of Petri nets and are a suitable trace semantics for BPMN models. The back-end component uses a combination of higher-order logic (HOL), a simply typed, functional language to write code and formal proofs, and SMT-LIB [16], a standardised first-order logic language to interface with many SAT and satisfiable modulo theory (SMT) solvers. In our work, we use the SMT solver Z3. This hybrid implementation allows us to:

- Choose between a HOL computation (either directly in the theorem prover Isabelle or in one of the functional languages Isabelle can generate) and SMT solvers; and
- Use Isabelle to formally verify our back-end using HOL and applying formally proven theorems to it.

To communicate between the front-end and the back-end, we use a straightforward JSON format describing a clinical guideline in terms of its LES semantics. We annotate conflicts in the same format to highlight them and visualise them. The different elements of the framework are described in the following sections.

⁴ See for instance <http://www.drugbank.ca>.

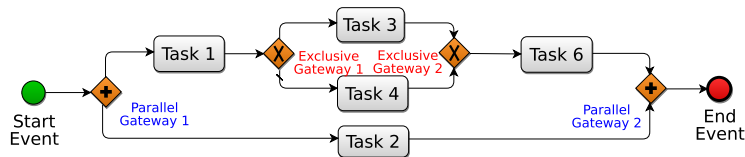


Fig. 4. A simple BPMN example.

3. The Front-end module

As we have discussed in the previous section, the notation used to capture guidelines is not adequate and lacks the precision required for automation. It is important to capture guidelines using precise domain-specific modelling languages or general-purpose languages such as BPMN or UML Activity Diagrams [14], as this will make automated reasoning feasible. Although none of these languages has been proven to be perfectly compatible with clinical pathway process modelling, BPMN is regarded as a promising notation amongst process-oriented modelling languages. Hashemian and Abidi analysed various techniques and picked out the main features in a comparison of BPMN with other approaches [22]. BPMN is a natural choice for our purposes as well, because it benefits from a broad acceptance within research and industry, the notation is fairly expressive and simple to use, and it is compatible with various existing process modelling tools [22,12]. It is hence straightforward to interface with this format and its intended semantics.

The input to our front-end module consists of one or more BPMN files in XML format. The module produces a JSON file, in a specific format, describing the corresponding LES used by the back-end component (cf. Section 4). It also produces a graphical representation of the LES which, with feedback from the back-end, can be used to highlight conflicts between the different CGs and identify the best treatment combination across CGs. Alternatively, we can also show the same information in a different format if required. One possibility is for this intermediate LES representation to be converted back onto a BPMN model with the best path highlighted there, a simple enumeration of problems and preferred alternatives, or other more suitable visualisation mechanisms for clinicians. Note that this last step is outside the scope of the present paper and requires clinical evaluation.

As an example of BPMN notation, Fig. 4 shows a simple BPMN model of a process starting with a parallel gateway (which splits the control flow) where the top branch involves Task 1 followed by a choice (exclusive gateway) between Task 3 or Task 4, followed by a merge and by Task 6. Task 2, in the lower branch, is executed in parallel, and the process ends after merging back the branches in the parallel gateway. No conditions are shown here but could be present in an exclusive gateway.

In the clinical context, a BPMN task corresponds to a step in a medical treatment, such as Step 2 for the hypertension clinical guideline in Fig. 2. Underlying Step 2 there is a selection of possible medications that can be given to a patient. This information must be documented and kept for each node in a BPMN model.

The front-end component works by extracting knowledge from a given guideline in accordance to a specific *ontology*, and by mapping elements of this ontology into LES concepts. We introduce the ontology in Section 3.1, recall LES briefly in Section 3.2, and give a description of the BPMN to LES mapping in Section 3.3.

3.1. BPMN_{CP} ontology

We follow the approach taken in [22] where ontologies – representing entities, their properties and mutual relationships – are established for both BPMN and CGs. A *clinical guideline ontology* describes a number of constructs for clinical guidelines. These constructs include *decision-based branching* (a decision is taken on which alternative to follow), *concurrent-based branching* (multiple steps for the treatment are executed concurrently), *activity flows* (ordering between two consecutive elements in a guideline), and so on. The following are the elements in a work-flow of a clinical guideline:

- *Action_Step* denotes the occurrence of a clinical action, which can be further specialised as *assessment_step*, *diagnostic_step*, *treatment_step*, *schedule_step* and *notification_step*. Most common action types in our examples are *diagnostic_step* (example: “HbA1c measurement”) and *treatment_step* (example: “Calcium-channel blocker”).
- *Decision_Step*, as the name suggests, is a step which requires a decision to be taken. This usually involves a choice of which path to follow and what the next *action_step* to be carried out within the guideline should be. This includes two main cases: *provider_decision_step* involves a decision by the provider (agent) responsible for medical activities, and *system_decision_step*, which encompasses decisions by the system such as a health information system. We focus on the *provider_decision_step* (example: “if metformin is contraindicated or not tolerated”) for our purposes.
- *Route_Step* is an activity flow which includes *Branch_Step*, indicating branching (that is, multiple steps being performed in parallel), and *Synchronization_Step* which joins the branches back into one flow.
- *Data_Element* is an attribute of all classes. All the domain-specific information which we need will be described in this element.

The $BPMN_{CG}$ ontology is a subset of the whole BPMN ontology to focus on representing clinical guidelines. Each element has common attributes of `id` and `name`, together with elements of incoming (except `Start_Event`) and outgoing (except `End_Event`) flow description.

`Event` denotes the occurrence of an action step at a particular moment of time and enables a process to progress. There are three types of events:

- `Start_Event` is the unique initial event of the process. It has no incoming flow and a unique outgoing flow.
- `End_Event` specifies the end (event) of the process. The event has one incoming flow and no outgoing flow. It represents the end point of the clinical guideline.
- `Intermediate_Event` is any other kind of event which can be triggered by a certain cause such as a timer or a notification.

Notice that our assumption of a unique start event implies that if a treatment has several possible starting points (e.g., the first line medication in the treatment of hypertension depends on age and ethnicity of the patient and different paths are used to describe this), it can be modelled by a `Start_Event` followed by a `Branch_step` and the following action steps.

`Activity` is a general term for something done within a process. Every action taken by any actor or participant is seen as an atomic activity, and hence an `Action_Step` is mapped as an activity.

- `Task` represents a job to be performed in the process which can be regarded as a basic and indivisible unit of work. A `Task` is a kind of `Action_Step`.
- `Sub_Process` (example: “Insulin-based treatments”) is any decomposable activity. The internal details are modelled using `Flow_Object` [15] such as `Gateway`, `Sequence_Flow` (introduced below), `Activity` and `Event`. In other words, it contains a valid BPMN diagram inside it. Hence, a `Sub_Process` is a modularised `Activity`.

A `Gateway` is used for depicting a diverging or a converging flow. Typically, diverging gateways allow a single input and multiple outputs while converging gateways take multiple inputs and generate a single output.

- `Exclusive_Gateway` controls mutually exclusive flows. Exactly one of the subsequent paths is valid for diverging branches. So `Exclusive_Gateway` (diverging) appears where a `Decision_Step` exists. `Exclusive_Gateway` (converging) may appear in case subsequent common elements exist for the branches split by an earlier divergence. Such a form of control flow can be used to model, for example, the branching arising between metformin-tolerant and intolerant patients in the guideline for diabetes (see Fig. 1).
- `Parallel_Gateway` controls multiple tasks to be completed to proceed. All the following branches of the diverging `Parallel_Gateway` join by a converging `Parallel_Gateway` at some point, and to proceed on from the converging point the completion of all the incoming branches is a necessary condition. Thus, diverging and converging `Parallel_Gateway` represent `Branch_Step` and `Synchronization_Step` respectively. Such a form of control flow can be used to model situations in which there are no restrictions on the order in which certain steps are executed; this happens, for example, when reviews have to be conducted for several prescribed medications. An instance of this is given by the treatment of COPD (chronic obstructive pulmonary disease), where it is common to have a number of medications (including roflumilast and corticosteroids) to be reviewed in parallel.


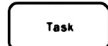

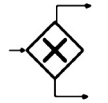
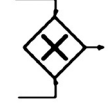
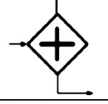
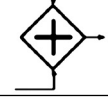


The ontology includes further elements such as `Connecting_Object` and `Data_Object` in addition to `Flow_Object` above.

- `Sequence_Flow` is a kind of `Connecting_Object`, representing the flow between a pair of other elements above. It is defined using `sourceRef` for indicating its source element and `targetRef` for indicating its target element. This represents the flow relation between each step in a clinical guideline.
- `Property` (example: “AGE”) is used to provide a data description to each element. It contains data but does not appear on the (visual part of the) BPMN diagram.

In [22], `Property` is defined just to deliver or store any generic additional data. Even though BPMN provides the possibility of specifying name, type and value of variables in `Property`, we need to describe logical and arithmetic general formulas to be passed to the SMT solver. To this end, we chose to store such information in the `Documentation` field, specifying a dedicated format for the possible entries, and handling all the details to interface the final user with the back-end: for example, converting the user-friendly infix notation with the Polish notation used in SMT-LIB. Ontology relationships between clinical guidelines and $BPMN_{CG}$ are a subset, selected according to our goals, from the ones provided in [22], and are summed up in Table 1.

The complete information stored within a BPMN model is contained in the corresponding XML file. In the next sections, we introduce our model used for the semantics, namely labelled event structures (LES), and then define the equivalence relationship between the $BPMN_{CG}$ ontology and LES.

Table 1
Mapping between elements in the clinical guidelines and $BPMN_{CG}$.

Clinical Guideline	$BPMN_{CG}$	Notation
Beginning and End point of the clinical guideline	Start_Event, End_Event	
Treatment_Step Diagnostic_Step (Action_Step)	Task or Sub_Process	 
Decision_Step	Exclusive_Gateway (diverging)	
Merged steps of exclusive branches	Exclusive_Gateway (converging)	
Branch_Step	Parallel_Gateway (diverging)	
Synch_Step	Parallel_Gateway (converging)	
Data_Element	Property	
Flow between steps	Sequence_Flow	

3.2. Labelled event structures (LES)

In our work, the model we use to give a semantics to behavioural models and scenarios of execution is based on labelled (prime) event structures [36,27] or LES for short. This model can capture directly the main notions contained within $BPMN_{CG}$ models for guidelines as we have described in [6]. Our presentation of the model here is similar to what we have done in earlier work [6].

LES is a very simple model that describes the notions and concepts we need in a straightforward manner, which includes sequential, parallel and iterative behaviour (or the unfoldings thereof) [27,8]. A LES offer simple notions over sets of events to denote event occurrences together with binary relations for expressing causal dependency (*causality*) and nondeterminism (*conflict*). Causality implies a (partial) order among event occurrences, while conflict expresses how the occurrence of certain events excludes the occurrence of others (e.g., an event occurring in one branch of a diverging exclusive gateway excludes events in another branch). From the two relations defined over the set of events, a further relation is derived, namely the *concurrency* relation. Two events are concurrent if and only if they are completely unrelated, i.e., neither related by causality nor by conflict.

These relations have a natural correspondence to the constructs from our clinical guideline ontology and consequently our defined $BPMN_{CG}$. For instance, causality represents *activity flow* (a $BPMN_{CG}$ sequence flow object), conflict represents *decision-based branching* (events within different paths of a $BPMN_{CG}$ exclusive gateway) and concurrency represents *concurrent-based branching* (events within different paths of a $BPMN_{CG}$ parallel gateway). The correspondence of the notions is described in Section 3.3.

The formal definition of a LES is given as follows (cf. [27]).

Definition 1. An event structure is a triple $E = (Ev, \rightarrow^*, \#)$ where Ev is a set of events and $\rightarrow^*, \# \subseteq Ev \times Ev$ are binary relations called *causality* and *conflict*, respectively. Causality \rightarrow^* is a partial order. Conflict $\#$ is symmetric and irreflexive, and propagates over causality, i.e., $(e\#e' \wedge e' \rightarrow^* e'') \Rightarrow e\#e''$ for all $e, e', e'' \in Ev$. Two events $e, e' \in Ev$ are *concurrent*, written

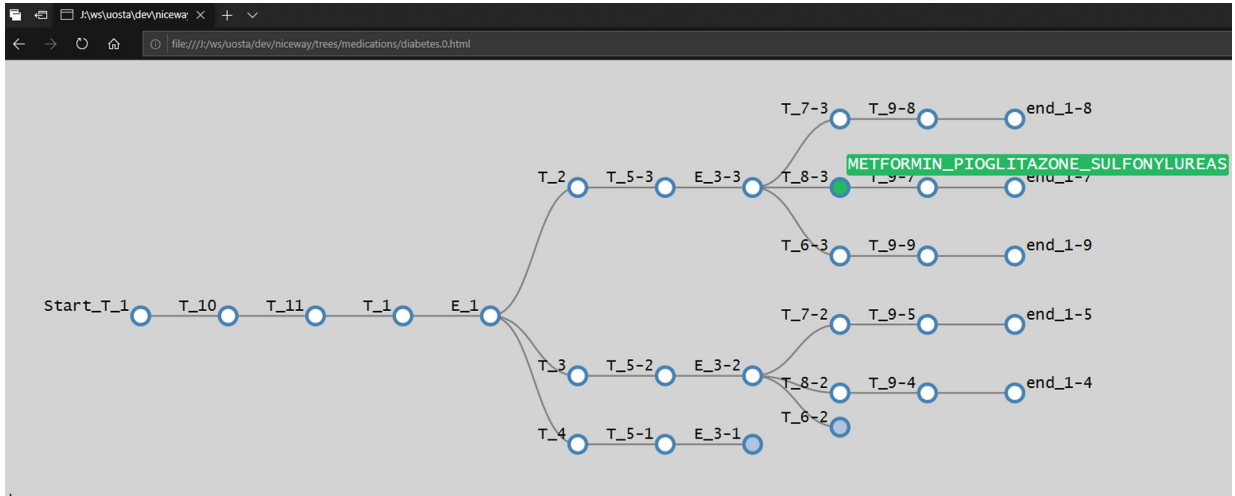


Fig. 5. Type 2 Diabetes LES when metformin is tolerated. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

$e \text{ co } e'$, iff $\neg(e \rightarrow^* e' \vee e' \rightarrow^* e \vee e\#e')$. Furthermore, $C \subseteq Ev$ is a *configuration* iff (1) C is *conflict-free*: $\forall e, e' \in C \neg(e\#e')$ and (2) *downward-closed*⁵: $e \in C$ and $e' \rightarrow^* e$ imply $e' \in C$. A maximal configuration is called a *trace of execution*.

In order to use event structures as defined above to provide a semantics for another representation, we need to establish a connection between the elements in the event structure and those of the associated syntactic model (in our case $BPMN_{CG}$). This is achieved by defining a labelling function over events, where the labels denote elements from the syntactic model. Let L be a given set of labels.

Definition 2. A *labelled event structure* is a pair $M = (E, \mu)$ over a set of labels L , where $E = (Ev, \rightarrow^*, \#)$ is an event structure, and μ is a labelling function such that $\mu : Ev \rightarrow 2^L$ maps each event onto a subset of elements of L .

In our case, the set of labels L denotes domain specific information as it appears in a CG model (given as a $BPMN_{CG}$), and we use it to either denote formulas (constraints over integer variables, e.g., $age \leq 55$, $y = 5$, or $HbA1c \leq 48$) or logical propositions expressing actions or patient conditions (e.g., perform blood test, metformin not tolerated, and so on).

Fig. 5 shows the interactive graphical output of the LES for type 2 diabetes. An event is coloured (e.g., E_{3-1}) if further details on the branch are currently hidden. These details can be shown again by double clicking on the event. The event label (such as associated medications) can be shown by passing the mouse over an event which shows the label and highlights the event (e.g., Metformin, Pioglitazone and Sulfonylureas are the medications associated to event T_{9-7}).

The figure was obtained by modelling the NICE pathway from Fig. 1 as a BPMN model (see Section 5 for more details), and then applying our front-end to automatically translate it into a LES. Our front-end is then able to visualise the formal model as shown, and the user can interact with the shown model as described. Fig. 6 shows the respective output for the hypertension CG that was partially described in Fig. 2. It shows the medications given at a particular stage in a path. In both cases, the model's visualisation is shown before we run a conflict and resolution check on the models. The result of such analysis would further highlight conflicts and suggest best paths accordingly. This will be shown later in Section 5.

3.3. From $BPMN_{CG}$ to LES

As mentioned before, the *ontology mapping* between $BPMN_{CG}$ and LES is straightforward since the notions and relations available in LES are very natural for the present purposes. The details of the mapping are as follows:

- *Start_Event*, *End_Event*, and *Task* elements are regarded as events in the event structure. In particular, a start event is a minimal event, and an end event is a maximal event (with respect to causality).
- *Sub_Process* is flattened by connecting its *Start_Event* and *End_Event* to the corresponding predecessor and successor events of the *Sub_Process*. The ordering of further elements within a *Sub_Process* is kept.

⁵ The terminology reflects the fact that the arrow symbol \rightarrow^* is commonly considered, by convention, to go from a greater event to a smaller one, according to the partial order it defines. In this paper, this convention has no particular significance.

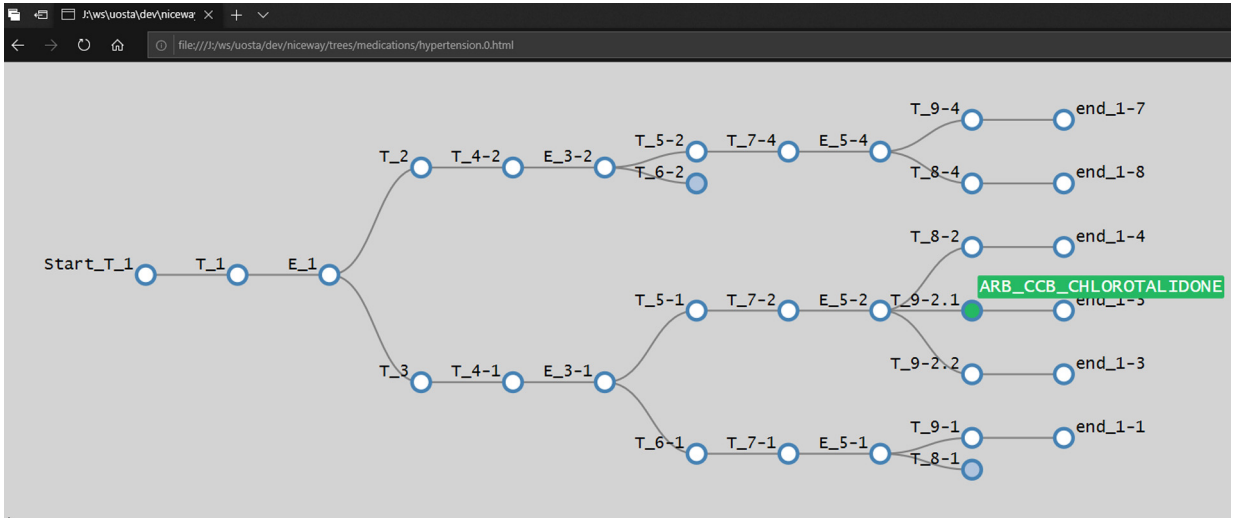


Fig. 6. Hypertension LES.

Table 2 Mapping between the *BPMN_{CG}* ontology and the event structure semantics.

<i>BPMN_{CG}</i>	Event Structures	Notation
Start_Event	Starting point of the event structure (minimal event)	ev_start
End_Event	End point of the event structure (maximal event)	ev_end
Task	Point existing part-way	ev_task
Exclusive_Gateway (diverging)	Diverging point with conflict notation	ev1
Exclusive_Gateway (converging)	Merging point and all the following events are copied	ev12
Parallel_Gateway (diverging)	Diverging point for parallel events	ev1
Parallel_Gateway (converging)	Merging point for parallel events	ev1
Data_Object	Event label	ev_task: label
Sequence_Flow	Causality between events	ev1 ev2

- A diverging *Exclusive_Gateway* is mapped to an event and all successor elements, which are indicated by the outgoing flows of the targeted diverging *Exclusive_Gateway*, are defined to be in conflict (given by our formal relation #).
- A converging *Exclusive_Gateway* is mapped to a set of events, one for each path leading to the exclusive gateway. The same effect ripples through each successor element after the *Exclusive_Gateway*.
- A diverging *Parallel_Gateway* is mapped to an event and all the successor elements connected to the diverging parallel gateway on different paths are associated to concurrent events.
- A converging *Parallel_Gateway* is regarded as a merging event in the event structure. If a pair (diverging-converging) of *Parallel_Gateways* includes any pair of *Exclusive_Gateways* in it, the converging *Parallel_Gateways* have multiple options for the incoming flow which follows the pair of *Exclusive_Gateways*. This process is repeated in case of further nested pairs.
- A *Sequence_Flow* element corresponds to a pair in the causality relation of the event structure.

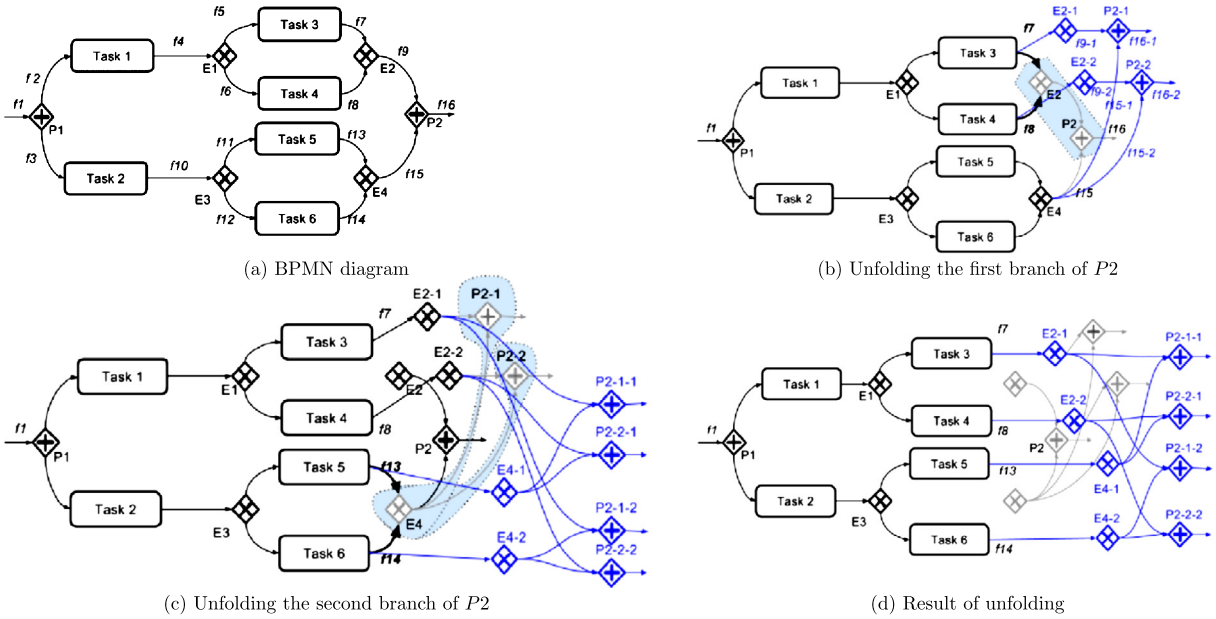


Fig. 7. Unfolding Exclusive_Gateway.

- Any Property used for domain-specific data description on the element is regarded as a label for the corresponding event.

We list the relationship between the BPMN ontology and semantics of event structures as described above in Table 2.

The mapping is fairly straightforward with the only concept that requires some further explanation being that of a converging Exclusive_Gateway. A LES is constrained by its relations, as described in Definition 1, and in particular the conflict relation # is symmetric, irreflexive and propagates over causality. To see the effect of this, assume two elements from different paths leading to a converging Exclusive_Gateway. These elements are associated to events, say event e_1 and e_2 , and these events are in conflict, i.e., $e_1 \# e_2$. If the converging Exclusive_Gateway corresponds to another event, say e_3 , then on the one side $e_1 \rightarrow^* e_3$ and on the other side $e_2 \rightarrow^* e_3$. The effect of the conflict propagation definition of # is that then e_3 would be in conflict with itself, which would violate the condition that # is irreflexive (no event can be in conflict with itself). For that reason, we have to duplicate event e_3 into e_{31} and e_{32} , where now $e_1 \rightarrow^* e_{31}$ and $e_2 \rightarrow^* e_{32}$, and consequently $e_{31} \# e_{32}$. All duplicated events are always in conflict. In addition, we have to duplicate all events associated to elements in the BPMN that appear after the converging Exclusive_Gateway. This can also be seen in the example of Fig. 5. Our automated BPMN to LES transformation deals with this by unfolding the paths of the original BPMN when exclusive gateways are present.

A converging Parallel_Gateway can be associated to only one event, provided there is no prior nesting of exclusive branches, since that preserves the relations in the LES. A case with nesting is shown in Fig. 7. It shows two diverging exclusive gateways (E1 and E3) executed in parallel.

We describe how the unfolding algorithm works. It starts by first unfolding one of them (say E1),⁶ thereby duplicating all the nodes following its corresponding converging gateway (E2). Since the causality relation is transitive, this duplication must be followed by a duplication also of the edges heading into all the nodes following the converging node (P2) corresponding to P1, since P2 has an incoming edge from E2. The results of this first duplication round are drawn in Fig. 7(b), where the nodes and the edges originating the duplication are greyed out. Now we have to iterate the process for the pair E3, E4, which must happen keeping in consideration the duplication spawned in the previous step (Fig. 7(c)). After unfolding, the structure goes through a procedure which trims the graph further. As seen in Fig. 7(d), there still remain nodes having no path to them or succeeding an invalid node: those are marked as not effective, and will be ignored when the final structure is generated. For example, E2 and E4 lost all the incoming flows to them and, therefore, all the following nodes are not effective. Finally, all the converging Exclusive_Gateway are not necessary after unfolding, because they do not join flows any longer, and, Start_Event and End_Event have no role after flattening either. Thus, all these nodes are removed from the final event structure.

⁶ This choice is made with no particular criterion. Introducing such a criterion, or unfolding E1 and E2 in parallel, might improve the algorithm, but has yet to be further investigated.

4. The hybrid HOL/SMT-LIB backend

Isabelle [31] is a theorem prover or proof assistant which provides a framework to accommodate logical systems (deductive rules, axioms), and compute the validity of logical deductions according to a given system. HOL is one of the most commonly used logical systems in Isabelle, and the resulting combination is called Isabelle/HOL. In the sequel, when we write Isabelle, we will mean Isabelle/HOL.

An SMT solver is a computer program designed to check the satisfiability of a set of formulas (known as assertions) expressed in first-order logic, where for instance arithmetic operations and comparison are understood, and additional relations and functions can be given a semantic meaning in order to make the problem satisfiable. Well-known SMT solvers include CVC3, CVC4, MathSAT, Yices, Z3 [16].

This section contains a description of how Isabelle and an SMT solver are used to provide the building blocks of our architecture: we will start by rendering into Isabelle code the definitions needed to introduce the notion of LES and the related mathematical tools; then, we will explain how to use the Isabelle code to obtain SMT code for LES.

Afterwards, we will show an example of an Isabelle theorem stating the correctness of the mathematics describing LES; this will be followed by the introduction of morphisms translating to/from Isabelle's higher order logic into SMT-LIB first-order logic, together with a *gateway theorem* to automatically transpose correctness of the Isabelle code to the SMT code generated by it. Moreover, we will show how to use the obtained SMT code to detect conflicts and their reasons. Finally, we will extend the notion of conflicts to a more realistic one, taking into account the temporal separation at which conflicting events happen. To do so, we will need to consider the order in which events happen, so that we can compute a function clock determining the time location of a given event, and then introduce a function f taking the clock separation of events to modulates the intensity of the conflict accordingly. How the function f performs this job is completely arbitrary: the section ends by suggesting some of the infinite possible f 's which can be used in practice.

In Isabelle, it is straightforward to define what an event structure is. This amounts to imposing the properties appearing in Definition 1 on the two relations \rightarrow^* (denoted below with Ca) and # (denoted below with Co).

```
abbreviation "IsLes Ca Co == (Reflex Ca & Antisym Ca &
Trans Ca & Irrefl Co & Sym Co & Propagation Co Ca)",
```

where the definitions of Reflex, Antisym, Trans correspond respectively to the properties of reflexivity, antisymmetry and transitivity (making up the notion of a partial order appearing in Definition 1), while Irrefl, Sym and Propagation correspond respectively to the properties irreflexivity, symmetry and propagation appearing in the same definition. These properties are all easily defined in Isabelle: therefore, we only show the definition of propagation:

```
abbreviation "Propagation Co Ca ==
(∀ x y z. ((Co x y & Ca y z) → Co x z))"
```

IsLes is a higher-order function returning whether its two arguments Ca and Co (each of which is a function of two arguments returning whether the corresponding events are in the relation which that function represents) form a valid LES. The definition of IsLes conforms to Definition 1 makes it possible to take advantage of the SMT code generator provided by Isabelle, and therefore to exploit SMT solvers' capabilities of checking whether a given model is a valid LES.

This will result in a series of assertions about Ca and Co, expressed in the first-order SMT-LIB format. For example, the assertion for the property of transitivity of a causality relation (automatically obtained from the HOL one above) will be:

```
(assert (forall ((x Ev) (y Ev) (z Ev))
(=> (ca1 x y) (=> (ca1 y z) (ca1 x z))))
```

SMT solvers are powerful enough to provide a useful aid to theorem provers' users, helping them to fill gaps in formal proofs by providing existing facts enabling the formal validity of each proof step. For these reasons, modern theorem provers, including Isabelle, provide SMT code generators. In this paper, however, we make a novel use of such a generator: we will not use the SMT generator for theorem proving, but rather to obtain formally verified SMT code for computing our solutions.

While Isabelle's built-in SMT-LIB generator is provided exclusively for proof searching (through Isabelle's tool *sledgehammer*[4]), it can be exploited to generate SMT-LIB code which can be fed to an SMT solver as Z3 to actually compute objects of interest to our present work. Therefore, a definition as `IsLes` acts as a gateway between HOL computations, SMT solver computations and theorem proving. This is useful, making it possible to switch to the computing paradigm (among HOL and SMT solving) which performs best or is best suitable, according to the kind of object to be computed; and, at the same time, to prove theorems about the correctness of both kinds of computations (see, for example, the theorem in Listing 1). Such a *gateway definition* has the advantage of being automatically transformable in SMT-LIB code by Isabelle's generator; however, in order to produce SMT code which is usable for our goals, such a definition must restrict itself to using only the mathematical object an SMT solver is aware of: functions, integer and real numbers, booleans and little else. An unwanted effect of this restriction is that this complicates proving Isabelle theorems about such definitions.

Let us exemplify this phenomenon concretely by considering, again, `IsLes`. Its definition contains basic properties of relations (reflexivity, symmetry, etc.), which we will need theorems about if we want to prove theorems about `IsLes`. Such theorems already exist in the Isabelle repository; however, these existing theorems make use of the usual set theoretic representation of a relation (i.e., a relation is exactly the set of ordered pairs of elements which are related). In contrast, as

is immediate looking at the definition of Propagation above, and for the reasons just explained, the relations appearing in `isLes` are represented as boolean functions over all the pairs, rather than sets: this is because SMT solvers are not aware of sets, in general. Our solution to this problem is to introduce a second, equivalent (as we will prove below) definition of LES, making use of the usual set-theoretical representation of relations:

```
abbreviation "isLes causality conflict ==
propagation conflict causality &
sym conflict & irrefl conflict & trans causality &
antisym causality & reflex causality"
```

It should be noted that this definition involves the same notions involved in `ISLES` definition, only expressed using set-theoretical constructions. This makes `isLes` more viable when computing some objects related to labelled event structures, for example the list of configurations:

```
abbreviation "configurations causality conflict ==
{X. X ∈ Pow (events causality) & extension causality X ⊆ X &
X ⊆ restriction conflict X}"
```

Such a compact (and computable) definition is possible exactly because set-theoretical constructs and notions (as \in , \subseteq and the power set) are now available, whilst things would have been much less direct, had we based our computations exclusively on `ISLES`. The auxiliary definitions involved in `configurations`, that is, `extension` and `restriction`, take advantage of set-theoretical concepts, too:

```
abbreviation "extension Ca X == (X ∪ (Ca-1 'X))"
abbreviation "restriction Co X == X - (Co 'X)".
```

As well as further set-theoretical operations (union and set difference), these definitions additionally take advantage of the functional higher-order operator of image of a set through a function (`'`).

Besides allowing expressive and direct definitions, `isLes` is also more suitable than `ISLES` to undergo correctness certifications by using theorem proving. Suppose, for example, that we want to be sure that the `configurations` set computed above returns indeed the correct set. This is certified by the following theorem: which reconnects our com-

```
theorem assumes "isLes causality conflict" shows
"config ∈ configurations causality conflict ↔
(isConflictFree conflict config &
isDownwardClosed causality config)",
```

Listing 1: The Isabelle theorem stating the correctness of configurations.

putable definition of configuration with that contained in the original Definition 1. The latter is expressed in terms of conflict-freeness and downward-closeness, whose straightforward rendition into Isabelle/HOL is omitted here. We proved this kind of theorem for the crucial objects we need to compute, and any additional computation we will add in the future will have the possibility to undergo the same type of validation. Such theorems add confidence that not only the derived objects (as `configurations`) do indeed compute the right thing, but also that `isLes` defines indeed the correct concept of labelled event structures. This is particularly important, because such correctness carries over also to our SMT-LIB code, once we prove (as we did) the equivalence theorem between `isLes` and the gateway definition `ISLES`:

```
theorem
"ISLES causality conflict ↔
(isLes (pred2set causality) (pred2set conflict))".
```

This theorem implies that, as long as we trust the correctness of Isabelle's SMT-LIB generator, all the correctness theorems that we prove for `isLes` carry over to the SMT-LIB code we pass to the SMT solver.

In order to state the theorem, we needed to implement HOL functions to pass from set theory (used for `isLes`) to first-order logic (used for `ISLES`):

```
abbreviation "set2pred R == (curry (λ a. a ∈ R))"
abbreviation "pred2set R == Collect (split R)",
```

where we used the standard operations of lambda abstraction and currying, together with the operators `split`, which is the inverse of currying (i.e., given a map yielding for each value of a variable x a function over a variable y , `split` returns a function over pairs (x,y)) and `Collect`, which returns the set of all values making its argument true.

To complete the equivalence theorem, we have proved Isabelle theorems to show that these converters are actually one-to-one, so that the worlds of the two definitions `isLes` and `ISLES` are indeed isomorphic:

```
lemma lm01: "set2pred ∘ pred2set = id" [... ]
lemma lm02: "pred2set ∘ set2pred = id" [... ],
```

where \circ is functional composition and id is the identity function (here, as for the previous formal theorems, we omitted the proofs). The two lemmas above formally express the fact that the SMT code (generated by first-order Isabelle definitions such as `IsLes`) and the Isabelle definitions about which we prove our correctness theorems are actually the same “up to isomorphisms” (where the isomorphisms are `set2pred` and its inverse `pred2set`), and therefore we can freely choose in which of the two worlds to work, according to the needs. This freedom also applies to the two lemmas above themselves: stating and proving them in the first-order language of SMT solvers would have been quite harder than in higher-order logic, where their Isabelle proofs present no particular challenges.

We emphasise that, having proved the correctness of the generated SMT code, we can use both Isabelle and SMT solvers to detect inconsistencies in a given BPMN structure, and to propose solutions. This is a different problem than finding conflicts between different structures, and is addressed elsewhere [9].

We have given an example of object `configurations` which is conveniently computed in HOL (and whose correctness can be proven as exemplified by the theorem in Listing 1) ; now, we give an instance of a computation which is best performed using an SMT solver, thereby motivating our hybrid approach.

The task is to find possible conflicts between the labels of events in different clinical pathways.

This extra checking step makes it possible to have a formal verification that the original CG was correctly captured into a BPMN, and that the front-end produced consistent output. Moreover, such formal specifications can be used for further computations regarding the composition of several CGs, as detailed in [9]. We also use Isabelle and SMT solvers to detect inconsistencies in the LESs generated automatically from two or more (single disease) BPMN models and propose resolutions. An inconsistency arises from the event labels of different models, such as when the actions associated to the events should not occur together (for instance, two medications usually administered for different treatments are harmful when combined). In finding conflicts between different event labels, one must also consider the arithmetic constraints associated to the events and check whether they are pairwise unsatisfiable. Such constraints can denote critical bounds concerning a blood value, medication dosage, and so on.

Consider a simple example where an event `T_4` has a label given by `z>1`, which would be translated into `(assert (> z 1))` for the back-end. Assume that in a different CG we have an event `Q_8` with a label given by `z<-10` which would generate a similar assertion, passing it along with the former to the SMT solver. The SMT solver would detect a conflict and output the pairs of events in conflict (in this case, `T_4` and `Q_8`), along with the reason of the conflict (the two irreconcilable arithmetic statements about the variable `z`). This is done for every possible combination of events, and passed back to the front-end for a graphical representation of the conflicts.

We proceed to enrich this semantics framework with a more powerful notion of conflict. In practice, the adverse effect of interactions between treatments such as the one derived from different medications that should not be given together is often expressed by a degree of severity (as we will see in Section 7) but in addition, rather than to highlight every single event with a conflict, we only care about the actual events that realistically denote concurrent treatment actions at a certain moment of time. We will see how to reconcile our LES semantics with this consideration, using an SMT solver. In doing so, we add a notion of time to our LES semantics, and compute the concurrent execution of distinct CGs minimising the degree of conflict.

Formally, we consider the directed acyclic graph (DAG) structure corresponding to an event structure partial order, described by its covering relation G (i.e., the node n_2 is a child of the node n_1 iff $(n_1, n_2) \in G$). This correspondence is always possible for a finite partial order. We also weigh its edges through a map $w : G \mapsto \mathbb{N}$. (G_1, \dots, G_m) is a given list of such DAGs, each representing a CG unfolding. The nodes are the single steps of each CG, the edges describe how they are causally connected, while the weights model the time elapsed between the occurrence of subsequent steps. The idea is that we want to take into account the temporal separation between the occurrence of two conflicting prescriptions (e.g., Insulin for diabetes versus Eprosartan for hypertension) for a more realistic resolution of such conflicts, by computing a global score taking into account all the possible conflicts between the pair of executed nodes, along with the time separating them. To this end, we need to consider all possible paths from the source of each G_i to each of its sinks, and pick exactly one such path for each i , while maximising the global score [3]. In the context of our application domain, that of composition of clinical guidelines, each graph represents a CG, the source of each graph is the first step in that CG, each path represents the sequence of steps chosen, among the possible ones, for each CG, and the score is a suitable metric (described in the sequel) to guide this choice. This metric combines, via an arbitrary function f which can be changed (and therefore has the role of a parameter), the interaction associated to single medications with their actual time distance in the chosen CG execution.

We hence have two problems: how to describe the notions of DAG and of path in SMT; and how to compute the score, taking time into account. We will face them in the sequel of this section. First, however, we note that there we will not show the very SMT code for our implementation. This is because SMT code is typically little readable by humans, due to the first-order logic underpinning it, which features a limited number of native notions and structures; so that, for example, any computation involving elementary operations on sets (e.g., union, intersection, cartesian product, domain, range, ...) has to be rewritten because sets are not directly available in first-order logic and must be represented as predicates which, anyway, cannot be directly performed operations on, because they are not first-class objects in the logic. Furthermore, to increase efficiency, usually a number of transformations are applied to the code making it even less readable: for example, a universally quantified assertion over a finite type is often replaced by multiple non-quantified assertions, each for one el-

ement of the type (quantifier elimination). Finally, SMT-LIB, the standard specifying a common language for SMT solvers [1], consistently employs polish notation, aggravating the problem on the human side.

Our solution to this expository problem is to write formulas close to the first-order logic language used by SMT solvers; for the sake of readability, however, we will employ some simplifications. In particular, we adopt infix notation instead of prefix notation, we use set-theoretical styling instead of predicates (e.g., writing $(j, k) \in G_i$ in lieu of $G_i j k$), we will use set-theoretical operations (e.g., union, intersection, cartesian product, domain, range, ...) in lieu of the corresponding first-order logic renditions, we will omit type specifications, and we will use the universal quantifier \forall even when in the actual code it has been eliminated.

We need to resolve two conceptually distinct problems: computing paths in each DAG and picking one for each DAG such that the overall score is maximum. For the first problem, there are well-known and efficient algorithms. However, rather than computing the two problems separately, we express the whole task as an SMT query and find a solution for it in one shot.

Despite this, we can easily distinguish between the SMT assertions expressing the first problem and those expressing the second.

For the first, remember that we are given the covering relation for each of the DAGs: therefore, we can also consider the relation resulting from their union (obtaining a single DAG with N nodes), and, similarly, consider as a solution the union of all the paths, one for each DAG. Such an overall path will be described by boolean variables $n_1 \dots n_N$, one for each node: each variable will describe whether that node is part of the solution path.

To begin with, the source of each G_i must be selected, meaning that the corresponding n_i must be true. After that, we need to impose that exactly one of its children is true. Similarly, for any of such children set to true, we impose that, in turn, exactly one of its children is true. This is repeated until a sink is reached (the node has no children). Besides doing that, we want to make sure that no other node is selected. Correspondingly, we generate, for the node n_i , the assertions

$$n_i \rightarrow \bigvee_{j|(i,j) \in G} \left(\bigwedge_{\substack{k \neq j, \\ k|(i,k) \in G}} (\neg n_k) \wedge n_j \right) \text{ and } \bigwedge_{(j,i) \in G} (\neg n_j) \rightarrow \neg n_i. \quad (1)$$

Let us now introduce the assertions describing the second of the two conceptually distinct problems mentioned above. Since we want a concurrency-aware notion of conflict, the first notion we need is that of temporal separation between nodes representing events. To this end, each node n_i will happen at some integer time denoted with clock_i . It is natural to impose that, whenever two nodes are subsequent in a path, their temporal separation is the one dictated by the weight of the corresponding edge, which is returned by the function w introduced above. Therefore, we introduce the following assertion for each i, j such that j is a child of i :

$$n_i \wedge n_j \rightarrow \text{clock}_j = \text{clock}_i + w(i, j).$$

The second notion we need, as anticipated previously in this section, is that of score. More precisely, since we want a representation combining the (desirable or undesirable) effects of isolated medications and the (desirable or undesirable) effects of their interactions, we define two kinds of score, measured by integer SMT variables score_i and $\text{score}_{i,j}$ respectively. The first are the absolute improvements given by the prescription associated to the step in node i , while the second are the scores generated by possible conflicts between the prescription associated to each of the event i and j .

We first zero out the scores for the unselected nodes:

$$\neg n_i \rightarrow \text{score}_{i,j} = 0 \wedge \text{score}_i = 0 \wedge \text{score}_{j,i} = 0.$$

The following assertion takes into account how the known interaction between two generic selected nodes is influenced by their time distance:

$$n_i \wedge n_j \rightarrow \text{score}_{i,j} = f(\text{interaction}_{i,j}, |\text{clock}_i - \text{clock}_j|), \quad (2)$$

where $\text{interaction}_{i,j}$ are constants obtained by looking up a database (see Fig. 3) to obtain the degree of conflict between the treatments in n_i and n_j . f is a known function which is used to specify how the interaction between different treatments and the elapsed time combine together. We finally assign the sum of the $\text{score}_{i,j}$ and of the score_i to a variable, and ask for an SMT solution to all the assertions for the n_i 's maximising the global score.

Let us dwell on the role of the function f , first noting that there is a substantial amount of freedom in choosing it, since functions are first-class objects in the first-order logic used by SMT solvers, and since the linear arithmetic operators offered by SMT solvers make it possible to express the mathematics sufficient to introduce a variety of interesting conflicts mechanics. We give just few examples of possibly interesting choices for f .

- If f is a function depending only on its first argument (refer to (2)), we obtain a way of “silencing” some conflicts according to specific needs.

- If $f(x, y)$ is of the form $x\theta(\tau - y)$, where θ is the Heaviside function, then we obtain a simple mechanism to take into account only interactions not exceeding the fixed time distance τ , which therefore acts as a threshold. We will focus on this kind of application in the simple examples of Sections 5 and 6.
- A slight elaboration on the previous point consists of allowing τ to be a function of x (i.e., of the first argument of f), rather than a constant. In this way, one can fine-tune the threshold to the particular medications.
- The mentioned fact that functions are first-class objects in the logic of SMT solvers implies, in particular, that f can have any finite number of arguments, so that we can add any to the two appearing in (2). The possibilities are countless: to mention only one, this makes possible to alter the interaction between medications along the course of time, for example to reproduce drug tolerance, etc.

n_i , clock_i , score_i are, from a conceptual point of view, functions of the independent variable i ranging over all the N nodes. From a technical point of view, however, they are encoded as several SMT variables (one for each possible $i \in \{1, \dots, N\}$). This is because this choice results in better SMT performance. As a general phenomenon, of which the aspect just mentioned is an instance, we found that choosing the right representation for data can make big differences with respect to efficiency in finding a solution. One advantage of our approach is that such variations between different SMT representations can be undergone with confidence, because our SMT code is always linked to formally verified Isabelle/HOL code. This approach is quite general and can accommodate a number of variations and extension: this particular aspect is presented elsewhere [7]. The fact that we represent, in SMT, the functions mentioned above as separate variables each evaluating the function “pointwise” on each possible value of the independent argument (in our case, the node index i), has an important consequence: it makes it possible to reduce the number of universal quantifiers in our SMT code. This is especially relevant in the case of Assertions (1) to find paths: the most natural rendition in SMT of the concept of path is probably one relying on universal and existential quantifiers. Instead, we repeat Assertions (1) over all the relevant nodes. This improves performance, but comes at the price of expressing the concept of path in a less intuitive way. In contrast, the assertions above regarding clock_i , score_i and $\text{score}_{i,j}$ are themselves simple. A slight complication came from the fact that, while the uninterpreted SMT addition operator supports any finite number of operands, there is no direct, efficient way of summing over an intensionally defined set. Since our final goal is the maximisation of the sum of all score_i and $\text{score}_{i,j}$, we had to explicitly enumerate all the possible score_i and $\text{score}_{i,j}$ as operands of $+$, and ask Z3 to maximise the result. Overall, this way of formalising the concepts of path, scores and clock does not make use of quantifiers, rational or real arithmetic, arrays, uninterpreted sorts or functions; however, it needs integer arithmetic. Correspondingly, a sub-theory of the main SMT-LIB theory suffices when invoking the SMT solver to process our assertions, for example `QF_NIA` (quantifier-free integer arithmetic); one can even get away with smaller theories as `QF_LIA` (quantifier-free linear integer arithmetics) according to how the function f behaves. This helps the solver applying satisfiability techniques specialised to the given fragment of theory, and typically more efficient [16].

The freedom in choosing the behaviour of f , and the abstractness of time notion both contribute to the flexibility of our approach. This flexibility, however, is best exploited with the aid and expertise of a clinician or a domain specialist to help tune the mechanism to tailor it to the context at hand. For example:

1. Durations can be typically expressed using a wide range of time scales (minutes, hours, days) for different kinds of actions in a CG. Such durations will need to be converted into a suitable common time unity in order to be applied the present approach.
2. Typically, durations for CGs steps are not explicitly given, due to a variety of reasons including their variability across different contexts and patients: for example, a particular drug could be prescribed to be taken repeatedly at given time intervals, resulting in a graph with distinct nodes separated by edges with weights representing those time intervals. In such situations, we cannot rely on existing databases (although there is early-stage work to obtain such a database by mining available data [29]) to be fed to our back-end, and the domain expert has a key role in assigning suitable time weights to the model.

5. Example: diabetes and hypertension

To illustrate our approach, we take the guidelines for type 2 diabetes and hypertension described earlier in Section 2. These are two of the CGs considered in a medical study involving a hypothetical woman with five multimorbidities given in [11]. The corresponding BPMN diagrams are given in Figs. 8 and 9, respectively, while the corresponding event structures were shown in Figs. 5 and 6. Note that, the naming of the event structures is done automatically (e.g., `T_2`, `T_4-2`, and so on).

For the sake of this example, after obtaining all of the possible conflicts, we assigned them arbitrary severities, associated an arbitrary score to each event, and added possible weights of the edges (describing the time between each node), setting them to 1 for each edge. More precisely, referring to the entities introduced in Section 2, the values of score_i , where the subscript i is used to range over all the events, were set to 0 with the exceptions reported in Table 3.

Furthermore, the quantities $\text{interaction}_{i,j}$ (described in Section 2) were set to 0 except for the pairs (i, j) identifying conflicts as detected in Section 4, where that quantity was set to -1000 . Finally, the function f governing the overall

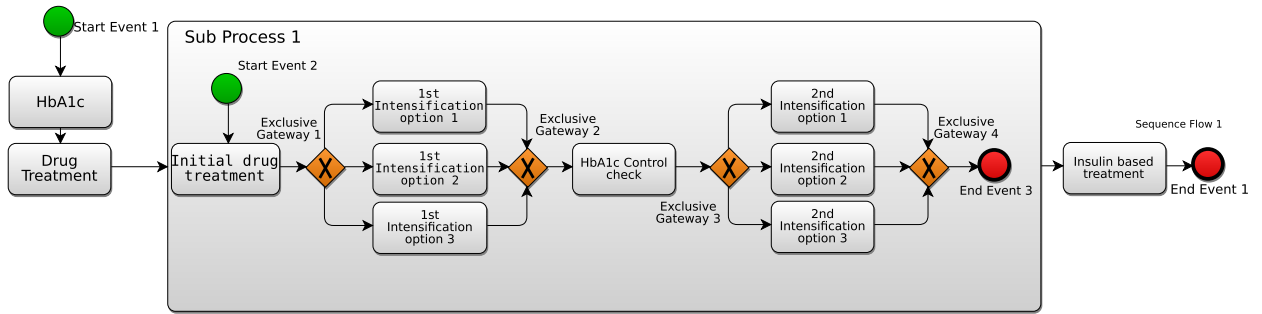


Fig. 8. Type 2 Diabetes BPMN when metformin is tolerated.

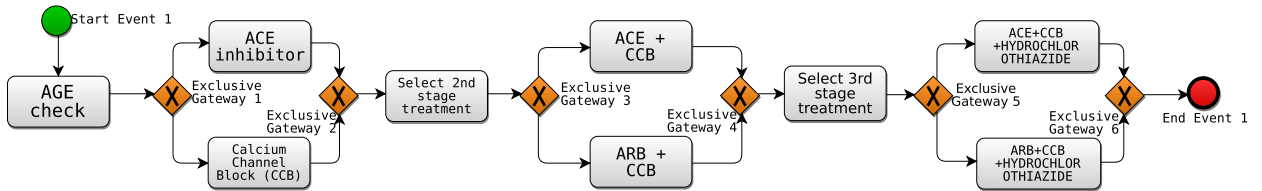


Fig. 9. Hypertension BPMN.

Table 3

Non-zero scores for events in the diabetes (left) and hypertension (right) CGs.

Diabetes		Hypertension	
events	score	events	score
E_1,T_9-4	1	E_5-4,T_6-2,T_8-2	1
E_3-1,T_4	2	T_9-2	2
T_9-2,E_3-2,T_9-6	3	T_7-1,T_9-1	3
E_3-3,T_5-1	4	E_5-1	4
Start_1	5	T_2	5
T_1,T_5-2,T_9-8	6	T_7-2	11
T_10	7		
T_11,T_5-3	8		
T_2	9		
T_6-1	10		
T_6-2	12		
T_6-3	14		
T_7-1	16		

impact of conflicts according to time was chosen to be a simple threshold function, yielding its first argument when its second argument is less than 9, and 0 otherwise.

The results are shown in Figs. 10 and 11 through screenshots of a javascript-based interface which can be visualised in any modern web browser. The interactive interface provides path colouring for optimal solutions (in orange in Figs. 10 and 11), tooltip to show the name of the medication associated to each event (in green in Figs. 10 and 11), and tree collapsing for irrelevant events (blue events as described earlier, and shown in Fig. 11).

The diagrams in Figs. 10 and 11 are meant to be a simple and intuitive representation of the solution found by the SMT solver. This solution is actually stored as a number of SMT-LIB definitions found by the SMT solver, defining the objects satisfying all the assertions describing the problem (see Section 4). For ease of use and clarity, the graphical interface does not convey all the particulars of the underlying solution: any detail of the latter, however, can be learned by directly evaluating the corresponding SMT definitions using any SMT solver. In this manner, one can investigate, for example, what is the corresponding score (i.e., the sum of all the $score_{i,j}$ and $score_k$ when the indices i, j, k range on all the nodes as explained in Section 4); which, for this case, is 75. Another query which can be evaluated is the list of the nodes conflicting with a given node. For the node h_T_9-2 , in the current example, this list is $d_T_3, d_T_7-1, d_T_7-2, d_T_7-3, d_T_8-1, d_T_8-2, d_T_8-3$. We note that the back-end automatically makes the name of each node unique, which, in the output above, has been attained by prefixing each node name by d (for diabetes) and h (for hypertension). One can also be interested in knowing conflicting labels between a pair of nodes: in the case of h_T_9-2 and d_T_3 , this is given by hydrochlorothiazide and sulfonyleureas; or in knowing the effective interaction (taking into account their time separation) of two nodes, which can be done by evaluating the function $score_{i,j}$ seen in Section 4 when i and j denote the nodes at hand: taking again h_T_9-2 and d_T_3 , we obtain 0, meaning that, for this simple example, the solver could find solutions such that the two nodes are

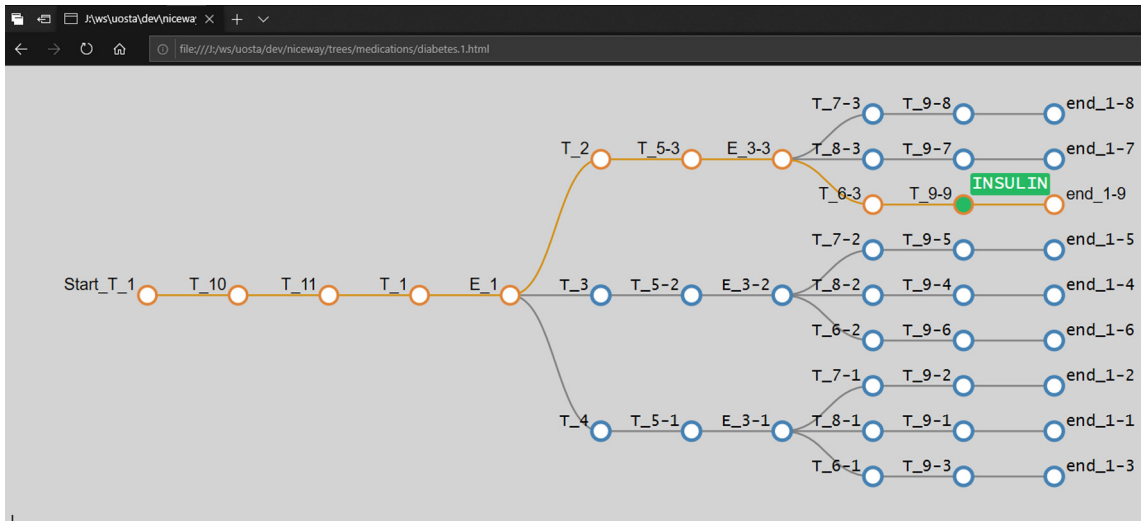


Fig. 10. Solution minimising conflicts in diabetes CG.

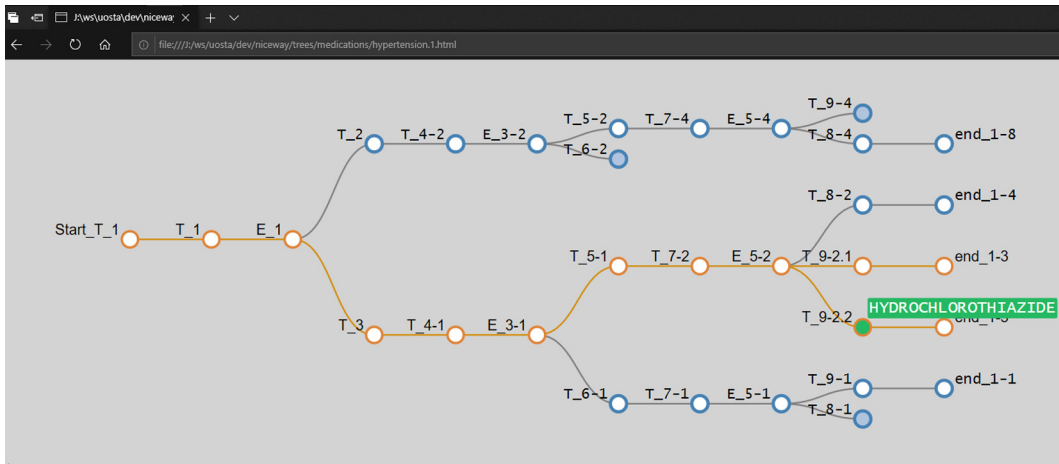


Fig. 11. Solution minimising conflicts in hypertension CG.

distant enough not to trigger conflicts. Finally, it could be interesting to know which selected node is happening at each clock tick, and which absolute score it has. The example is expanded in the next section.

6. Sub-optimal conflict resolutions

Section 4 introduced a way of dealing with the presence of inter-CG conflicts by the introduction of a metric to quantify them (through a quantity we called score) which allowed us to find a CG execution which is optimal according to this metric (i.e., it maximises the score). In practice, it can be of interest to also find other executions which are not the optimal ones, but are still good enough to be considered; they can, for example, provide fallback alternatives in case optimal ones must be discarded for some reason. E.g., because of some particular context, or because some condition is not captured by the formal CGs, or because the arbitrary attachment of score to single treatments make small differences in the overall score calculation insignificant. One technical problem, however, is that SMT solvers, in general, provide only one model: in our case, it is an optimal one, and in case of ties the solver breaks them arbitrarily. In other words, there is no direct way of querying the solver for other solutions. We introduce an automated technique allowing to overcome this limitation by obtaining, sorted by optimality, all solutions of our problem. Afterwards, we will apply this technique to obtain some of the best solutions for the example problem already considered in Section 5.

The idea is to first compute the optimal execution using the SMT code introduced in Section 4 (let us, in this section, refer to that code as the *base assertions*), then add assertions to the SMT solver's stack imposing that the next solution differs from the previous solution (along with all the previous assertions characterising the previous solution); iterating this scheme until the SMT solver returns *unsat* will give all the possible traces of executions by exhaustion. While the idea

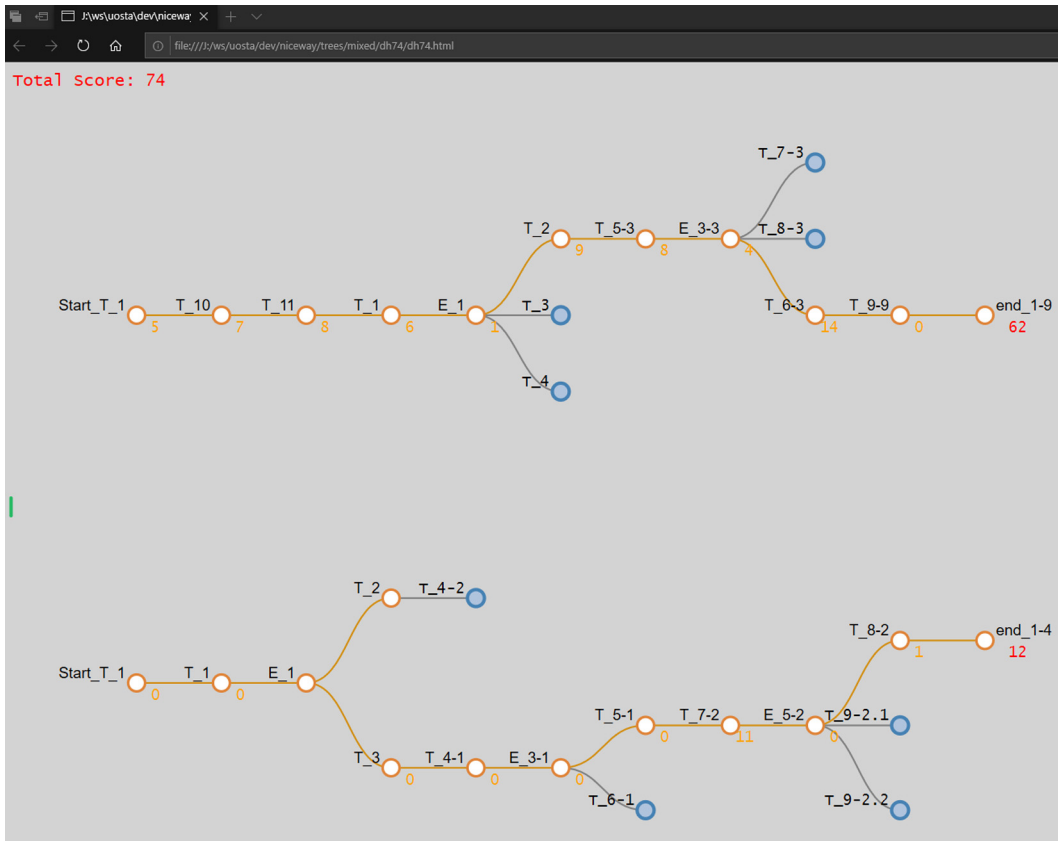


Fig. 12. Solution of score 74 (diabetes at top and hypertension at bottom).

is simple, there is no direct way in SMT-LIB of referring to previously found solutions in subsequent assertions.⁷ There are provisions to attain this in some APIs provided by some SMT solvers [3]; however, giving up SMT-LIB in favour of a particular API would mean no longer being able to apply Isabelle correctness proofs to SMT code as outlined in Section 4. Additionally, these APIs are dependent on the particular SMT solver used, while SMT-LIB has the advantage of preserving some freedom in the choice of the solver to run our SMT code on.

Our solution is to keep the SMT solver running as a background process, waiting for commands to be supplied through a UNIX named pipe (let us call it `smtInPipe`), and outputting its results to another named pipe (let us call it `smtOutPipe`). In parallel with this background process, a simple UNIX Bourne shell script loops analysing the output received on `smtOutPipe`: when the script sees `sat`, it requests the SMT solver the current solution through the SMT-LIB command (`get-model`), saves the solution on a cumulative file, formulates additional assertions requiring the next solution to be distinct from the current solutions and sends them to the solver through `smtInPipe`. This is iterated until the script sees `unsat` on `smtOutPipe`, or until some other condition is met by the last generated model.

The Bourne shell script (less than 20 lines, not reproduced here) parses the last output to extract the value of the overall score for the last solution, imposes the overall score for the next solution to be strictly lower, and triggers a new SMT search using (`check-sat`). This implementation is just a basic prototype of the idea using UNIX processes interacting via named pipes, and can be extended, for example by distinguishing solutions presenting exactly the same score value.

We applied this technique to the example of Section 5, and concluded that the best score value is 75, followed by 74 and 72. The output for the score value of 75, being the optimal solution, corresponds, as seen in Section 5, to Figs. 10 and 11. To appreciate the difference between the various solutions, the graphical interface can optionally display further details about the scores of each CG step, as shown in Figs. 12 and 13.

More precisely, Fig. 12 describes the solution of score 74; Fig. 13 does the same for the solution of score 72. Notice that the interactive interface only shows the individual score of each node (corresponding to the function $score_i$ seen in Section 4), not the interaction scores $score_{i,j}$; this is to avoid cluttering the graphical output. As explained in Section 5, it is always possible to query the SMT solver for details about the computed solution.

⁷ See the comment at https://stackoverflow.com/questions/11867611/z3py-checking-all-solutions-for-equation#comment15793591_11869410 from one of the lead developers of the Z3 SMT solver.

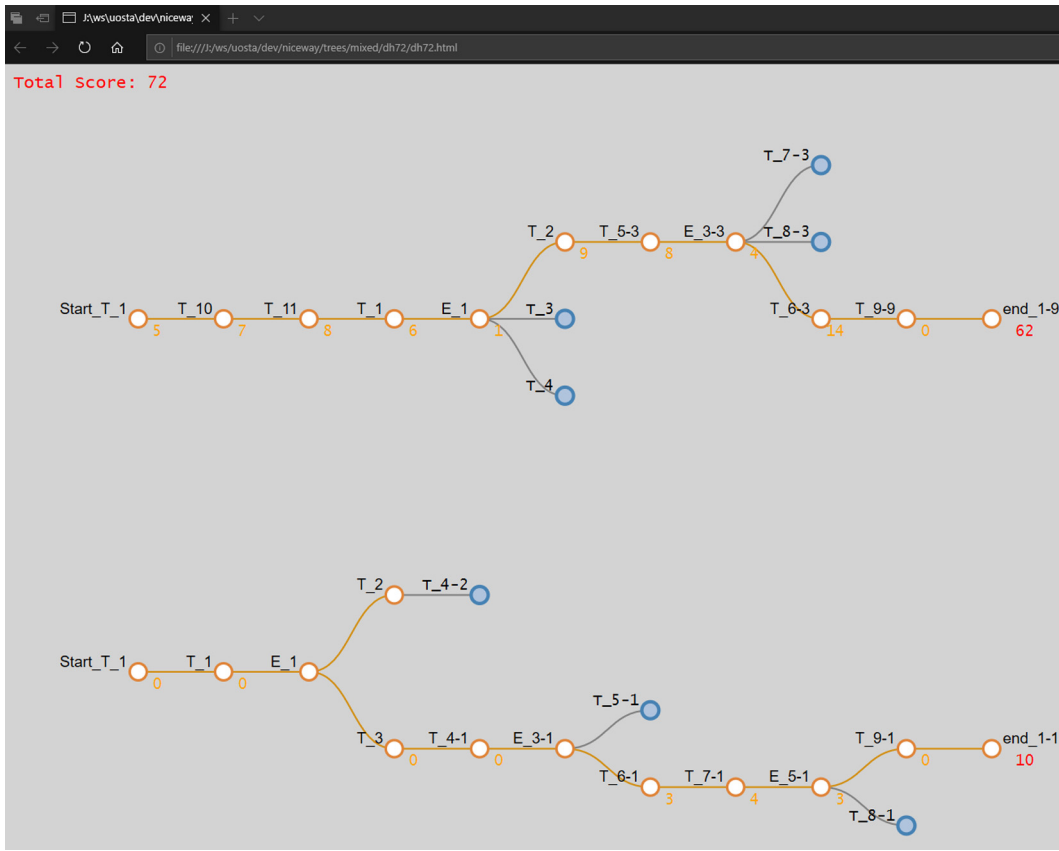


Fig. 13. Solution of score 72 (diabetes at top and hypertension at bottom).

7. Related work

We classify existing related work into three broad strains: papers dealing with the general problem of representing and computing BPMNs with effects potentially relevant (in a broad sense) for the goals of this paper, papers specifically addressing a problem in the same context (CGs automatic computation) and papers dealing with other general approaches (not specifically addressing CGs or involving BPMNs) also potentially relevant for the goals of this paper.

7.1. Generic BPMN approaches

BPMN is a widely adopted standard across industry and academia for a variety of problem domains, consequently there is also a range of work using BPMN models for computational goals some of which provide a formal semantics to BPMN. For example, [13] provides a BPMN semantics using Event-B, [37] defines such a semantics using the specification language PROMELA, and in [17] a natural, Petri nets-based semantics for BPMN is introduced. In turn, these semantics permit a range of different forms of automated analysis involving BPMNs possibly relevant to the main problem of this paper (combination of CGs). For example, once PROMELA models are obtained, access is granted (e.g. via the model checker SPIN [24]) to the study of livelocks, deadlocks and the verification of LTL (linear temporal logic) properties over a combination of BPMNs and, therefore, possibly, of CGs. Such a study seems suitable to the management of optimisation problems such as optimal allocation of resources and staff, and streamlining of procedures. A similar paper using BPMN for resource planning in the healthcare domain and using performance evaluation techniques for that purpose is given by [10]. The present paper could be considered as close in spirit to [17], since the semantics we adopted (event structures) have a well-known interpretation as unfoldings of Petri nets [30]; besides this foundational aspect, however, there is little in common here with the contents of [17]. We note that a formal model based on event structures as in our case serves our purpose of optimal path search more naturally. In [26], the authors look at medication conflicts specifically and use the SMT solver Z3 to find the conflicts and automatically suggest alternatives based on a notion of score associated to medications. We have addressed, in a previous paper [9], the detection of structural inconsistencies in single BPMN models, as opposed to the problem of combining multiple CGs faced in this paper. In that paper, the same underlying semantics (ES) is used, but in its treatment the computational roles of Isabelle and of the SMT solver differ slightly compared to the present approach: the `unsat-core` capability of the SMT solver is applied to single out events causing inconsistencies in the structure, and events are automatically added

to resolve this, through ad-hoc computable Isabelle definitions and the intervention of Isabelle-provided counterexample finders.

7.2. Approaches specific to the automatic processing of CGs

There are a number of groups working on the specific problem of automated processing of CGs, using a variety of semantics, formalisms and representations. The Performance Evaluation Process Algebra (PEPA) [23] formalism is combined with coloured stochastic Petri nets to automatically represent and process CGs in [38], which is however focused on the resource allocation problem, and hence also closer to the work in [10]. A further approach [21] presents a similar approach through the introduction of a PEPA specialisation called Clinical Pathway PEPA (CPP). While the context of the present paper is similar to that of [21] and [38], our goals are very different, since we focus on evidence-based guidelines for handling well-understood chronic conditions, which do not provide probabilities, therefore preventing the possibility of a probabilistic analysis.

However, the problem of automated detection and resolution of guideline conflicts in patients with multimorbidity is considered in other papers. Paper [34] adopts a data-enriched subset of BPMN (called BPMN+V) as a representation language for CGs, and an extension of Workflow Graphs as an execution-oriented semantics, together with coloured Petri nets as an analysis-oriented semantics. The latter allows one to detect conflicts upon combining CGs using space state analysis. On one hand, there is at least one clear advantage provided by the use of Petri nets compared to our approach, in that the slight complications arising from unfolding exclusive gateways (see Section 3.1) can be avoided; and this also facilitates, potentially, the treatment of cycles in the given CGs (although their treatment is not currently included in [34]). On the other hand, resorting to state space analysis gets problematic for large models which the authors in [34] acknowledge. The authors also suggest an extension of the proposed methods to allow the use of SAT/SMT solvers as a future development. Our approach combines theorem proving with (optimising) SMT solvers providing scalable and efficient solutions. Furthermore, the simplicity of our formal model (LES) is key and facilitates verification with our used theorem prover Isabelle.

In addition, [39] and [40] introduce an ad-hoc formal model called Transition-based Medical Recommendation (and extensions thereof) for the CG combination problem. They describe relations between actions and states using first-order logic, which are implemented using Web Semantics technologies. The fact that TMR is quite close to first-order logic permits a great flexibility, allowing to introduce a number of relations to closely describe actual clinical contexts. A disadvantage of that approach, however, is the absence of a notion of time separation between events which, in our case, is facilitated by the presence of an intermediate modelling mathematical structure such as an event structure.

More exhaustive overviews of existing approaches to the same problem can be found in the review papers [2], [19], [32].

7.3. Other approaches adopting representations or semantics possibly suitable to CGs combination problem

Finally, there is work not specifically targeted at the problem of combining CGs, and not based on the BPMN standard, but which provide formulations which could be possibly applied to such a problem. In our recent paper [5], we treated the problem of describing a complex system from partial specification, as an abstract problem formulated using the same mathematical model (event structures) used in the present paper. Compared to the present paper, however, that model is enriched with additional labelling functions to describe priority and duration of execution events, with a focus on a novel concept of dephasing between component executions. We approached the same problem in [7] where we used directed acyclic graphs (DAGs) as the underlying modelling structures. The focus in [7] is on the rendition of DAGs as formally verified SMT code. Furthermore, [5] and [7] respectively introduce general methods to obtain new correct-by-construction code for the relevant underlying models.

Similar problems concerned with composing concurrent models or executions are studied, among others, in [35] (where sequence diagrams are composed relying on SAT solvers), in [28] (which also focuses on sequence diagrams formalised as typed graphs), in [33] and in [41] (where UML class diagrams are automatically composed after translating them into Alloy code).

8. Conclusion and outlook

Multimorbidity is becoming increasingly common and is recognised as a major public health priority. Clinical guidelines chiefly address single diseases, and it is unpractical to produce multiple guidelines for all possible disease combinations. Instead, we require automated tools to aid clinicians and health care staff in producing recommendations that can help suit treatments to individual patients with complex needs. Our proposed automated framework in this paper makes a contribution towards addressing such challenges.

Our approach takes one or more BPMN models as input, and generates models in a simple intermediate mathematical representation (LES) suitable for relevant analysis, reasoning and several computations. The back-end of our framework features a novel way of integrating two radically different computing paradigms: higher-order logic (which can be used, via Isabelle/HOL, for formal correctness proofs and high-level computations) and SMT (which can be used, via very efficient SMT solvers, to find models satisfying a large number of constraints). The advantage of combining both in our framework is that we can opt for the best paradigm for every computational task at hand, and at the same time apply to both paradigms

the formal verification possibilities given by Isabelle/HOL. This is possible thanks to an original use of the SMT-LIB generator provided by Isabelle. We note that while its native, intended application is theorem proving, we use it to generate SMT code for different computational tasks. The SMT code that we generate directly from Isabelle, gives us an added bonus of correctness by construction for all the code employed in the back-end. The present paper extends [6] with new formal theorems in Section 4 showing the possibilities of the interplay between Isabelle and SMT solvers.

An important feature of our approach is the possibility to accommodate a notion of time between treatment steps, thereby allowing to specify, for example, how possible conflicts depend on their temporal separation and to compute how the execution of guidelines is affected. Furthermore, this paper introduces, as an extension to [6], a novel way of incrementally producing possible solutions sorted according to their score, without the need of restarting the SMT solver and therefore taking advantage of the computations which had been done earlier. Finally, the interactive, graphical user interface is a considerable extension of the work presented in [6], and gives a glimpse of how the work described in this paper could be presented to domain experts and final users.

There are a number of possible directions for future work, some of which we discuss briefly. First, we would like to add the possibility for clinicians to interact with the computations by adding or removing specific constraints in a dynamic fashion. Such an interaction is of the utmost importance, given the high degree of complexity and criticality of the task of passing from the medical knowledge to the formal model we propose: this translation has to be done by a clinician, in order to make sure that it leads to a medically sensible model. Second, it would be useful to find ways of accommodating indefinite cycles and non-terminating behaviours possibly featured in input BPMN models. At present, this would generate a (caped) number of unfoldings in our LES. Third, while the back-end features formal proofs certifying whether the event structure obtained from the front-end is valid or not, it would be useful to provide the front-end with a similar formal approach, certifying that the algorithm producing a LES from the given BPMN model is correct: this would make sure that errors possibly found in the LES model by the back-end are due to inconsistencies in the BPMN input, rather than to the conversion algorithm in the front-end. Work is under way to achieve this. Finally, as we have seen in the examples, the level of detail describing the solution provided by the SMT solver is not easy to be conveyed by a graphical interface. We want to explore further mechanisms to improve the usability of our approach in practice.

References

- [1] C. Barrett, A. Stump, C. Tinelli, The SMT-LIB standard: version 2.0, in: A. Gupta, D. Kroening (Eds.), *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories*, Edinburgh, UK, vol. 13, 2010, p. 14.
- [2] E. Bilici, G. Despotou, T.N. Arvanitis, January 2018. The use of computer-interpretable clinical guidelines to manage care complexities of patients with multimorbid conditions: a review. *Digital Health* 4.
- [3] N. Bjørner, A.-D. Phan, L. Fleckenstein, νz -an optimizing SMT solver, in: *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, in: LNCS, vol. 9035, Springer, 2015, pp. 194–199.
- [4] J.C. Blanchette, S. Böhme, L.C. Paulson, Extending Sledgehammer with SMT solvers, *J. Autom. Reason.* 51 (1) (2013) 109–128.
- [5] J. Bowles, M.B. Caminati, Correct composition of dephased behavioural models, in: *International Conference on Formal Aspects of Component Software*, in: LNCS, vol. 10487, Springer, September 2017, pp. 233–250.
- [6] J. Bowles, M.B. Caminati, S. Cha, An integrated framework for verifying multiple care pathways, in: *2017 International Symposium on Theoretical Aspects of Software Engineering, TASE, IEEE, Sept. 2017*, pp. 1–8.
- [7] J.K. Bowles, M.B. Caminati, A flexible approach for finding optimal paths with minimal conflicts, in: *International Conference on Formal Engineering Methods*, in: LNCS, vol. 10610, Springer, 2017, pp. 209–225.
- [8] J.K.F. Bowles, Decomposing interactions, in: *International Conference on Algebraic Methodology and Software Technology*, in: LNCS, vol. 4019, Springer, 2006, pp. 189–203.
- [9] J.K.F. Bowles, M.B. Caminati, Mind the gap: addressing behavioural inconsistencies with formal methods, in: *2016 23rd Asia-Pacific Software Engineering Conference, APSEC, IEEE Computer Society, 2016*, pp. 313–320.
- [10] J.K.F. Bowles, R.M. Czekster, T. Webber, Annotated BPMN models for optimised healthcare resource planning, in: *Software Technologies: Applications and Foundations*, in: LNCS, vol. 11176, Springer, 2018, pp. 146–162.
- [11] C.M. Boyd, J. Darer, C. Boulton, L.P. Fried, L. Boulton, A.W. Wu, Clinical practice guidelines and quality of care for older patients with multiple comorbid diseases: implications for pay for performance, *JAMA* 294 (6) (2005) 716–724.
- [12] R. Braun, H. Schlieter, M. Burwitz, W. Esswein, BPMN4CP: design and implementation of a BPMN extension for clinical pathways, in: *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on, IEEE, 2014*, pp. 9–16.
- [13] J.W. Bryans, W. Wei, Formal analysis of BPMN models using event-B, in: *International Workshop on Formal Methods for Industrial Critical Systems*, in: LNCS, vol. 6371, Springer, 2010, pp. 33–49.
- [14] M. Burwitz, H. Schlieter, W. Esswein, Modeling clinical pathways-design and application of a domain-specific modeling language, in: *Wirtschaftsinformatik, 2013*, p. 83.
- [15] M. Chinosi, A. Trombetta, BPMN: an introduction to the standard, *Comput. Stand. Interfaces* 34 (1) (2012) 124–134.
- [16] D.R. Cok, D. Déharbe, T. Weber, The 2014 SMT competition, *J. Satisf. Boolean Model. Comput.* 9 (2016) 207–242.
- [17] R.M. Dijkman, M. Dumas, C. Ouyang, Formal Semantics and Analysis of BPMN Process Models Using Petri Nets, *Tech. Rep.*, Queensland University of Technology, 2007.
- [18] R.M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, *Inf. Softw. Technol.* 50 (12) (2008) 1281–1294.
- [19] P. Fraccaro, M.A. Casteleiro, J. Ainsworth, I. Buchan, Adoption of clinical decision support in multimorbidity: a systematic review, *JMIR Med. Inform.* 3 (1) (2015).
- [20] S. Government, Polypharmacy guidance, in: *Scottish Government Model of Care Polypharmacy Working Group, 2nd edition, March 2015*.
- [21] R. Han, X. Yang, A. Rowe, Y. Guo, Formal modelling and performance analysis of clinical pathway, in: *Proceedings of the NETTAB 2011 Workshop Focused on Clinical Bioinformatics, NETTAB'11, 2011*, pp. 66–73.
- [22] N. Hashemian, S.S.R. Abidi, Modeling clinical workflows using business process modeling notation, in: *Computer-Based Medical Systems (CBMS), 2012 25th International Symposium on Computer-Based Medical Systems, IEEE, 2012*, pp. 1–4.
- [23] J. Hillston, *A Compositional Approach to Performance Modelling, Distinguished Dissertations in Computer Science*, vol. 12, Cambridge University Press, 2005.

- [24] G.J. Holzmann, The model checker SPIN, *IEEE Trans. Softw. Eng.* 23 (5) (1997) 279.
- [25] L. Hughes, M.E.T. McMurdo, B. Guthrie, Guidelines for people not for diseases: the challenges of applying UK clinical guidelines to people with multimorbidity, *Age Ageing* 42 (2013) 62–69.
- [26] A. Kovalov, J. Bowles, Avoiding medication conflicts for patients with multimorbidities, in: *12th International Conference on Integrated Formal Methods*, in: LNCS, vol. 9681, Springer, 2016, pp. 376–392.
- [27] J. Küster-Filipe, Modelling concurrent interactions, *Theor. Comput. Sci.* 351 (2) (2006) 203–220.
- [28] H. Liang, Z. Diskin, J. Dingel, E. Posse, A general approach for scenario integration, in: *MoDELS, 2008*, LNCS, vol. 5301, Springer, 2008, pp. 204–218.
- [29] F.-r. Lin, S.-c. Chou, S.-m. Pan, Y.-m. Chen, Mining time dependency patterns in clinical pathways, *Int. J. Med. Inform.* 62 (1) (2001) 11–25.
- [30] M. Nielsen, G.D. Plotkin, G. Winskel, Petri nets, event structures and domains, part I, *Theor. Comput. Sci.* 13 (1981) 85–108.
- [31] T. Nipkow, L.C. Paulson, M. Wenzel, Isabelle/HOL: A Proof Assistant for Higher-Order Logic, Springer-Verlag, London, UK, 2002.
- [32] D. Riaño, W. Ortega, Computer technologies to integrate medical treatments to manage multimorbidity, *J. Biomed. Inform.* 75 (2017) 1–13.
- [33] J. Rubin, M. Chechik, S.M. Easterbrook, Declarative approach for model composition, in: *Proceedings of the 2008 International Workshop on Models in Software Engineering, MiSE '08*, ACM, New York, NY, USA, 2008, pp. 7–14.
- [34] P. Weber, J.B.F. Filho, B. Bordbar, M. Lee, I. Litchfield, R. Backman, Automated conflict detection between medical care pathways, *J. Softw. Evol. Process* 30 (7) (October 2017).
- [35] M. Widl, A. Biere, P. Brosch, U. Egly, M. Heule, G. Kappel, M. Seidl, H. Tompits, Guided merging of sequence diagrams, in: *SLE 2012*, in: LNCS, vol. 7745, Springer, 2013, pp. 164–183.
- [36] G. Winskel, M. Nielsen, Models for concurrency, in: S. Abramsky, D. Gabbay, T. Maibaum (Eds.), *Handbook of Logic in Computer Science*, in: *Semantic Modelling*, vol. 4, Oxford Science Publications, 1995, pp. 1–148, Ch. 1.
- [37] S. Yamasathien, W. Vatanawood, An approach to construct formal model of business process model from BPMN workflow patterns, in: *Digital Information and Communication Technology and Its Applications (DICTAP), 2014 Fourth International Conference on*, IEEE, 2014, pp. 211–215.
- [38] X. Yang, R. Han, Y. Guo, J. Bradley, B. Cox, R. Dickinson, R. Kitney, Modelling and performance analysis of clinical pathways using the stochastic process algebra PEPA, *BMC Bioinform.* 13 (14) (2012) 1.
- [39] V. Zamborlini, M. Da Silveira, C. Pruski, A. ten Teije, E. Geleijn, M. van der Leeden, M. Stuiver, F. van Harmelen, Analyzing interactions on combining multiple clinical guidelines, *Artif. Intell. Med.* 81 (2017) 78–93.
- [40] V. Zamborlini, R. Hoekstra, M. da Silveira, C. Pruski, A. ten Teije, F. van Harmelen, A conceptual model for detecting interactions among medical recommendations in clinical guidelines, in: K. Janowicz, S. Schlobach, P. Lambrix, E. Hyvönen (Eds.), *Knowledge Engineering and Knowledge Management*, Springer International Publishing, Cham, 2014, pp. 591–606.
- [41] D. Zhang, S. Li, X. Liu, An approach for model composition and verification, in: *NCM 2009*, IEEE Computer Society Press, 2009, pp. 1102–1107.