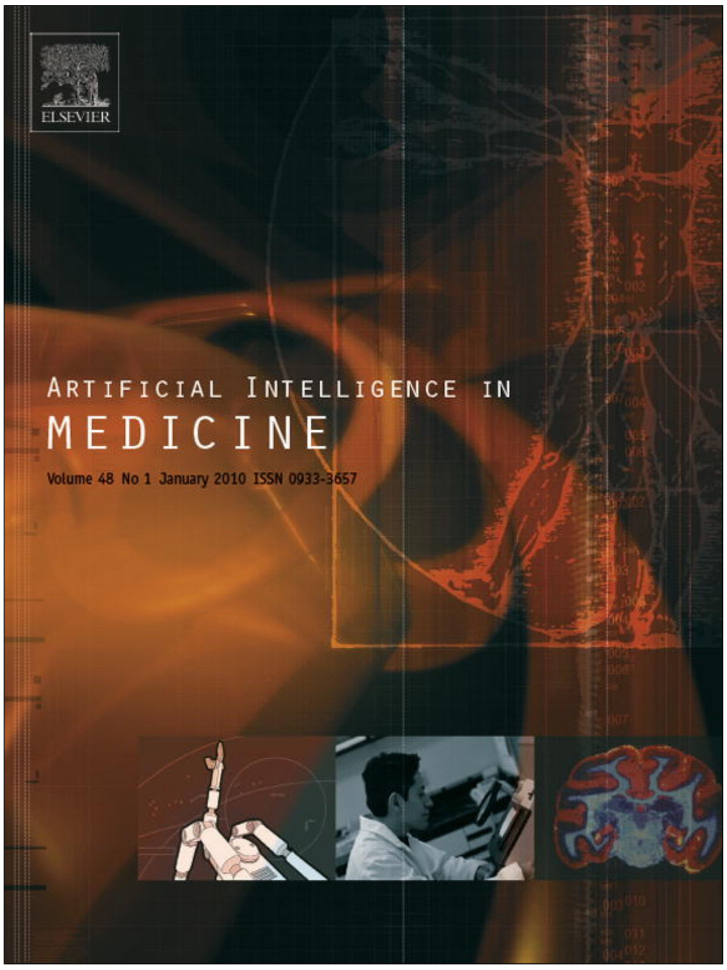

ELSEVIER
ARTIFICIAL INTELLIGENCE IN
MEDICINE
Volume 48 No 1 January 2010 ISSN 0933-3657

Artificial Intelligence in Medicine 48 (2010) 1–19

Contents lists available at ScienceDirect

Artificial Intelligence in Medicine

journal homepage: www.elsevier.com/locate/aiim

Adopting model checking techniques for clinical guidelines verification

Alessio Bottrighi [a], Laura Giordano [a], Gianpaolo Molino [b], Stefania Montani [a], Paolo Terenziani [a,*], Mauro Torchio [b]

[a] *Dipartimento di Informatica, Università del Piemonte Orientale "Amedeo Avogadro", Viale Teresa Michel 11, 15121 Alessandria, Italy*
[b] *Azienda Ospedaliera San Giovanni Battista, corso Bramante 88, 10126 Torino, Italy*

ARTICLE INFO

Article history:
Received 11 October 2007
Received in revised form 14 September 2009
Accepted 14 September 2009

Keywords:
Clinical guidelines
Model checking
Verification

ABSTRACT

Objectives: Clinical guidelines (GLs) are assuming a major role in the medical area, in order to grant the quality of the medical assistance and to optimize medical treatments within healthcare organizations. The verification of properties of the GL (e.g., the verification of GL correctness with respect to several criteria) is a demanding task, which may be enhanced through the adoption of advanced Artificial Intelligence techniques. In this paper, we propose a general and flexible approach to address such a task.
Methods and materials: Our approach to GL verification is based on the integration of a computerized GL management system with a model-checker. We propose a general methodology, and we instantiate it by loosely coupling GLARE, our system for acquiring, representing and executing GLs, with the model-checker SPIN.
Results: We have carried out an in-depth analysis of the types of properties that can be effectively verified using our approach, and we have completed an overview of the usefulness of the verification task at the different stages of the GL life-cycle. In particular, experimentation on a GL for ischemic stroke has shown that the automatic verification of properties in the model checking approach is able to discover inconsistencies in the GL that cannot be detected in advance by hand.
Conclusion: Our approach thus represents a further step in the direction of general and flexible automated GL verification, which also meets usability requirements.

1. Introduction

Clinical guidelines (GLs) can be defined as a means for specifying the "best" clinical procedures and for standardizing them. The adoption of GLs, by supporting physicians in their decision making and diagnosing activities, may provide crucial advantages, both in individual-based health care, and in the overall service offered by a health care organization. In particular, it has been shown [1] that GLs can improve the quality of patient care, reduce variations in quality of care, and reduce costs.

These observations justify the increasing number of GLs which have been defined in the last decade, covering a large spectrum of diseases and medical procedures. However, the effort in defining and disseminating GLs has not always been coupled by a parallel effort in guaranteeing their "quality" [2]: despite the fact that GLs are issued by recognized experts' committees, they might be ambiguous or incomplete [3], or even inconsistent.

The need for GL quality *verification* is thus clearly emerging. As we will show in this paper, computer-based approaches can provide crucial advantages in this context. The research community in Artificial Intelligence (AI) in medicine and in medical decision making, which is very active in the definition of computerized systems and projects for managing GLs (see e.g. the systems Asbru [4], EON [5], GEM [6], GLARE [7–8], GLIF [9], GUIDE [10], PROforma [11], and the collections [12–14]), has recently started to consider this issue.

Nevertheless, the verification capabilities available in the conventional computerized GL management systems in the literature are usually rather limited and only recently this limitation has led to the development of proposals for guideline automatic verification. Let us first analyse the limitations of conventional computerized GL management systems. In many cases, such systems do associate only very specific and ad hoc inferential mechanisms to the knowledge represented in the guideline. For instance, Asbru [15] and GLARE [16] adopt temporal-reasoning algorithms for temporal consistency checking, useful both for GL acquisition, and for simulation purposes. In GLARE, costs and resources required by the various GL actions can be collected, and the "admissible" paths in the GL (e.g. paths not exceeding a prefixed cost) can be identified on the basis of this result. Several systems [17] apply some controls for checking the well-formedness of the acquired GL, e.g. as regards name and range

* Corresponding author. Tel.: +39 0131360174; fax: +39 0131360198.
E-mail address: terenz@mfn.unipmn.it (P. Terenziani).

0933-3657/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.artmed.2009.09.003

checking of the actions of a GL and of its attributes (which must match specific standards), or as regards the adherence to several logical design criteria, such as the fact that alternative arcs may only stem from decisions.

However, two major drawbacks of the conventional approach can be outlined as follows (for a more detailed discussion see Section 2.2):

(i) every class of properties to be checked, for every GL (possibly with the exceptions of purely syntactical properties), requires the definition of an ad hoc verification software module. The analysis of an additional class of properties thus requires an additional effort by the programmers who are in charge of verification;
(ii) the verification process is not conceived as a flexible and incremental one: all properties to be verified must be known a priori, in order to let the verification software modules be developed before the verification process starts. Additional relevant classes of properties suggested by the already obtained results cannot be easily taken into account (due to the issue discussed in point (i)).

More generality and flexibility are therefore needed in guideline property verification. Generality is one of the main achievements of the theorem provers and model checkers developed within the automatic verification community [18]. Therefore, integration between the "physician-oriented" way of coping with clinical guidelines supported by the guideline management systems on one side, and the generality of verification techniques, on the other side, can provide fruitful results. Such an integration has started to be explored only quite recently within the Medical Informatics community. The adoption of theorem proving techniques has been first proposed within the Protocure European project starting in 2003 [2,19]. As an alternative of the theorem proving methodology, the adoption of model checking techniques has been first proposed few years later in the Protocure project [20] and in our GLARE project [21–23], mainly motivated by the simplicity and efficiency of model checking techniques with respect to the theorem proving approach [24]. In this paper we elaborate on the ideas first sketched in [21–23], extending and systematizing such an initial proposal (as discussed in Section 8, where we also explore in-depth the main differences between our approach and Protocure's one).

In particular, our paper focuses, on one side, on the knowledge representation and methodological issues (which are typically the main interest of *AI researchers*), and, on the other side, on usability issues (which are more interesting from the medical point of view) by analyzing which properties of the guidelines can be verified, and when. More specifically, the paper main contributions are the following:

(i) first, as a motivation for our approach, we propose an in-depth analysis of when the verification capabilities we provide can be used within the GL life-cycle, and a general overview of the different types of properties that can be verified (i.e., what can be verified);
(ii) second, we provide a general methodology to integrate verification capabilities within a GL management system. Specifically, we propose a modular approach in which a computerized GL management system is loosely coupled with a model-checker via a translator, which maps any GL expressed in the formalism of the computerized GL management system into the formalism of the model-checker. In such a way, the advantages of adopting a GL management system from one side, and a general-purpose model-checker on the other side are retained and combined. In particular, once the mapping has been defined, any class of properties that can be formalized in the logic of the model-checker can be easily verified, without requiring the definition of a new verification software module from scratch. This obviously facilitates a real interaction between the physician examining the GL and the system itself. Thanks to its modularity, such an approach can be easily implemented, since it does not require any modification to either the computerized GL management system or the model-checker;
(iii) third, we show how such a general approach can be instantiated. Although our proposal is mostly application-independent, as a proof of concept, we are currently integrating within the system GLARE [7] a verification tool which models a GL in Promela, the specification language of the model-checker SPIN [25], and verifies the GL properties to be checked by formalizing them as Linear time Temporal Logic (LTL) formulas. In particular, one of the contributions of the work relies in the analysis of how a GL can be represented in a process-based language such as Promela;
(iv) fourth, we refine the discussion about the different types of properties proposed in item (i), showing how they can be expressed using LTL. We also propose an application to the verification of the guideline about ischemic stroke as a concrete example.

The paper is organized as follows. In Section 2, we introduce our general goals and methodological choices. In Section 3 we summarize the main features characterizing GLARE, which will be needed to present our verification approach, and in Section 4, we briefly introduce the model-checker SPIN. In Section 5, we specifically present our implementation of model checking for verification in GLARE. In Section 6, we show several properties than can be checked during the GL life-cycle, classified as in Section 2. Section 7 contains a more extensive verification example, conducted on a real world GL. Section 8 is devoted to comparisons with related work. Finally, Section 9 contains our concluding remarks and future research directions.

2. General goals and methodology

In this section, we first show the advantages of adopting property verification throughout the computerized GL life-cycle, and then introduce our methodological approach to GL verification.

2.1. Using verification throughout the computerized GL life-cycle

It is important to recognize that, in the computerized GL life-cycle, different phases can be distinguished, and different actors play an important role. Specifically, we single out three main phases (namely (1) design and acquisition, (2) contextualization, and (3) execution), and we highlight how verification can be fruitfully exploited in each phase. As a result of such an analysis, different classes of properties are identified. Such classes will be further on elaborated, discussed and exemplified in Section 6.

2.1.1. Design and acquisition

GLs are usually defined by a national or international committee of specialists, and can be acquired into a computer-based system, usually through a cooperation between some specialists and some knowledge engineers. In such a phase, verification through model checking is useful in order to take into account at least two different classes of properties, namely structural properties and medical validity properties. In particular:

(i) *Structural properties* concern the existence of the appropriate clinical requirements. These properties regard the actions, conditions and paths of actions in the GL considered "per se", without any reference to the specific context of execution and to the specific patients on which the GL will be applied, and are relevant in order to ensure the appropriate management of any patients.

(ii) *Medical validity properties* concern both the exclusion of dangerous treatments and the inclusion of the most appropriate treatments for the considered class of patients. These properties are relevant in order to ensure best practice.

Both classes of properties are verified during the acquisition phase, in which both medical experts and knowledge engineers are usually involved. Specifically, medical experts can identify the structural and validity properties that are relevant for the GL under consideration, and knowledge engineers can formulate and run the corresponding verifications, reporting the results to the experts. In case the checks show that a desired property does not hold, the domain experts should identify the appropriate corrections to the GL, which will be modified accordingly, in cooperation with the knowledge engineers.

2.1.2. Contextualization

Once a GL has been defined and acquired (e.g., by a national or international committee), it has to be applied to several different local structures (e.g., hospitals). Unfortunately, in several cases, the original GL is too "general" to be applied on any specific environment. For instance, depending on the local availability of resources, certain actions of a general GL cannot be executed in specific contexts (e.g., small hospitals). A phase of contextualization is thus usually needed: when a new GL is introduced in a hospital, the medical personnel can use (possibly in cooperation with knowledge engineers) verification in order to identify which resources the GL (or specific paths of the GL itself) requires. Specifically:

(iii) *Contextualization properties* concern the resources needed for the GL execution and can be checked to adapt the GL to locally available resources.

The results of such verifications can be used for modifying the original GL, or for improving the hospital resources, in order to conform the hospital to the GL requirements (to grant the best practice). In the last case, the intervention of administrator personnel is also necessary.

2.1.3. Execution

Finally, the acquired and contextualized GLs are used in clinical practice. In such a case, a specific user-physician selects and applies a specific GL to a specific patient. Verification is a crucial support also in such a phase:

(iv) *Properties concerning the application of a GL to a specific patient* allow to check which are the best actions (as indicated in the GL) to be executed on the patient at hand, on the basis of the patient's status and symptoms; they also allow to check whether the GL (or some specific path of it) contains the specific actions which the user-physician expects to be necessary for the patient at hand.

In other words, during the execution phase, verification can be mostly used by physicians in order to check the applicability of the GL to the specific patient at hand (instead of checking, e.g., the "abstract" consistency and correctness of the GL itself). In this case, verification can be performed directly by user-physicians (since, unlikely in the other two phases, knowledge engineers are not involved at execution time). Of course, we are well aware that the direct use of a temporal logic (LTL in our approach, CTL in Protocure's one [20]) by a physician might be problematic. However, the analysis of the "relevant" properties we address in Section 6 can be seen as a first step in the direction of managing such an issue, since it can be helpful for designing a user-friendly interface, allowing users (not expert in LTL) to express at least the most "relevant" queries in a more natural way (for instance, by providing them with a menu listing a pre-compiled set of patterns). The development of such an (graphical) interface is a major goal of our future work, as discussed in Section 9.

It is also worth stressing that the distinction we have highlighted in this section regards the (time of) use of the verification facilities. Of course, the same (or similar) properties can be verified at different stages during the GL life-cycle. For example, the very same property checking the suitability of a (part of a) GL to cope with patients showing a given set of symptoms can be used both (i) during the acquisition phase, to check the eligibility of the given GL to treat a given class of patients, and (ii) during the execution phase, to check the applicability of the GL to treat the specific patient at hand. Moreover, different physicians may also want to verify conflicting properties on the same GL, motivated by different reasons, at different stages of the GL life-cycle.

2.2. Methodology: loosely coupling computerized GL management systems and model checkers

Most computerized GL management systems developed by the Medical Informatics community provide physician-oriented formalisms to represent medical knowledge, as well as user-friendly specialized facilities to acquire and consult GL, and to execute them on specific patients. Usually, such facilities are enriched with user-friendly interfaces, to allow systems to be used also by physicians non-expert in computer systems. However, as discussed in the introductory section, computerized GL management systems usually do not provide general-purpose facilities to express and verify properties about the GLs being acquired.

On the other hand, most model checkers developed by the automatic verification community are general-purpose engines which, taken in input any formula in a given logical formalism, and a formal description of the model (in the formal language provided by the model-checker), verify whether the input formula holds or not in the given model. Although, in principle, clinical GLs could be directly modelled using the language of a model-checker, this is practically unfeasible, since languages provided by model checkers are general-purpose, complex, and therefore quite far from being suited to represent GLs in the form in which physicians want to manage them.

One of the main goals of our approach is therefore that of preserving the physician-oriented environments provided by computerized GL management systems, improving them with the general-purpose facilities provided by model checkers.

The general approach we propose is to exploit the capabilities of a model-checker by loosely integrating it with a computerized GL management system. Such a loose integration can be provided by defining a module for the automatic translation of any GL which is represented in the computerized GL management system format into the corresponding GL represented into the model-checker input language format. Of course, the translator is language-specific (for instance, a specific translator is needed to translate GLARE GLs into SPIN's language Promela; a different one would be needed if Asbru GLs were taken into account), and "one-way" (in the sense that translating from the model-checker language into the one of the GL management system has no practical usefulness in this context). Given such a module, physicians can operate using the GL management system to acquire and execute GLs, and take advantage of the model-checker in order to verify any property (that can be expressed in the model-checker property language) on any of the system GLs. The overall process is shown in Fig. 1. Given a computerized GL management system, several guidelines ($GL_1, \ldots, GL_n$ in Fig. 1) can

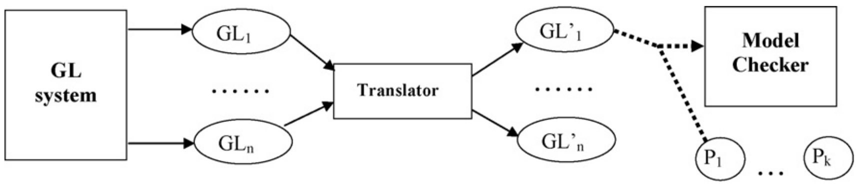


Figure 1. Our loosely coupled approach to deal with GL verification using a model-checker.

be acquired. The translator module can be applied in order to automatically translate such guidelines into the formalism used by the model-checker to represent the domain (so that GL_1', . . ., GL_n' can be obtained, as shown in Fig. 1). This is the initial situation, starting from which different users may check (for different purposes—see Section 6) different types of properties about the GL (e.g., P_1, . . ., P_k in Fig. 1). To check a new property P on a guideline GL the user has just to express P into the model-checker property language, and to invoke the model-checker on P and GL′ (i.e., the representation of GL in the language of the model-checker). For instance, in Fig. 1, the property P_1 is verified concerning the guideline GL_1. The output for the user will be a positive answer (if the property P_1 holds in the guideline GL_1) or a counter-example (if P_1 does not hold). It is important to stress that the properties need not to be defined a priori: the user can directly express a new property and ask the model-checker to verify it.

Such a methodology can be contrasted with the one currently used by many computerized GL management systems in the literature, which is schematized in Fig. 2. In conventional approaches, a specific "ad hoc" software module is used in order to check a class of properties about GLs (possibly with the exception of purely syntactical properties). Instead of expressing the property to be checked and invoking the model-checker (as shown in Fig. 2), in this case the user directly invokes the specialized module which has been devised in order to verify the property s/he is interested in (for instance, in Fig. 2, the "P_1 module" has been invoked in order to check the property P_1 on the guideline GL_1). Notice that, in such a case, the properties to be checked must be fixed a priori, and an ad hoc module must have been implemented for each one of them.

The general methodology we propose in Fig. 1 is much more general and advantageous, since:

(i) it does not require the development of many different software modules (one for each class of properties): the translator (e.g., from GLARE to Promela) is built once and for all, and after that any property (expressible in the model-checker property language) can be checked on any GL. This approach is less costly than the "ad hoc" approach. Suppose that a new class of properties has to be verified on a given GL. In the "ad hoc" approach, programmers must be involved, in order to implement the proper software module. Only after the algorithm is available, the user can use it in order to check the desired property on the given GL. On the other hand, in our approach, no intervention of programmers is needed: the user (knowledge engineer and/or physician) can directly express the new property in the language of the model-checker, and invoke it to do the automatic checking;

(ii) it is more flexible. Specifically, in the "ad hoc" approaches, designers are called to foresee "a priori" which are the classes of properties to be checked, in order to let programmers produce the proper software modules. Only the properties for which algorithms have been devised and implemented can be checked by users. On the other hand, in our approach, the set of properties to be verified has not to be defined a priori: users can directly check any property (provided that it can be expressed in the language of the model-checker);

(iii) it supports flexible and incremental sessions of verification. This is a direct consequence of issues (i) and (ii) above. In certain cases (e.g., in the phase of acquisition of a new GL), the incremental verification of its properties may be very important. For instance, during design/acquisition, the experts building a GL may want to verify the eligibility conditions of the GL being built, i.e., to check what is the typology of patients that the GL is able to deal with. Such a check can be usually performed through a sequence of different checks, each one depending on the results of the previous ones, and aiming at properly defining and restricting the class of eligible patients. Since each check depends on the result of previous ones, all approaches based on a pre-defined menu of properties are usually damned to fail (since there is usually no way of identifying a priori all the properties to be checked in such an interactive session of work). On the other hand, incremental sessions of work are fully supported by our approach, since the only restriction we impose is that the properties need to be expressible in the model-checker language.

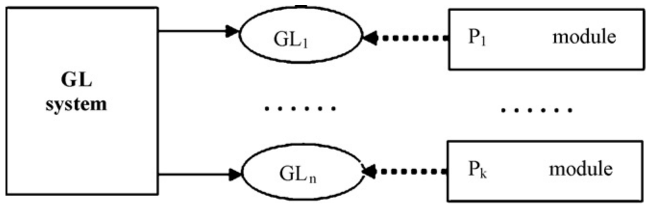


Figure 2. Using "ad hoc" algorithms to deal with GL verification.

In the rest of the paper we will describe a specific instantiation of the general architecture introduced in this section, by reporting our integration of GLARE with SPIN. In particular, we will focus on the representation of GLARE GLs in Promela (SPIN input language) and we will show how Promela constructs can be suitably exploited in order to model GL executions. Finally, we will discuss some details of the automatic translation process we have devised. Such a translation, on the basis of the general principles highlighted in this section, converts any GL represented in GLARE formalism into the corresponding Promela program. However, we insist on the generality of our methodology, which can be applied also to other computerized GL management systems and model checkers.

3. Computerized clinical guidelines: the GLARE approach

In this section, we briefly summarize the main features of GLARE, which will be needed to present our verification approach. GLARE is a domain-independent prototypical system for acquiring, representing and executing GLs. GLARE has been built within a 10-year project with Azienda Ospedaliera San Giovanni Battista in Turin, and which has been successfully tested in different domains, including ischemic stroke, bladder cancer, reflux esophagitis, and heart failure [7,16].

Despite in this section we refer to the characteristics of our system, it is worth noting that GLARE shares many choices with several different approaches to computerized GL management systems [17]. In particular, the basic philosophy underlying the architectural design of the systems described in the literature is based on the assumption that knowledge in the GLs is independent of its use (e.g., decision support, evaluation etc.), so that it is convenient to distinguish between the problem of acquiring and representing GLs and the problem of "using" them (e.g., "executing" acquired GLs on specific patients). In accordance with this approach, in GLARE we have defined two clearly distinguished modules, one for acquisition and one for execution.

Moreover, the GL representation primitives adopted by the systems described in the literature may differ for several details, but if we look at them at a more abstract level, we can identify a few skeletal concepts, which have been embedded in GLARE as well. The GLARE formalism, in particular, does not add many additional representation primitives to these fundamental skeletal concepts, but, as well as PROforma [11], it is based on an essential – and simple – set of constructs, meant to balance as much as possible the trade-off between expressiveness and complexity.

Section 3.1 will address GLARE architectural choices, while Section 3.2 will introduce the representation primitives.

3.1. System architecture

GLARE's acquisition module, as well as the acquisition modules of several other systems, embeds:

(i) a graphical interface, which supports primitives for drawing the control information in the GLs, and for acquiring the internal properties of the objects (see Section 3.2);
(ii) facilities for browsing the GLs.

GLARE also implements various forms of automatic consistency checking (e.g. temporal consistency checking via the proposal of advanced AI techniques [16]).

The GLs managed by the acquisition module are stored in a database.

On the other hand, the execution module executes an acquired GL for a specific patient, taking into account the patient's data, automatically retrieved from the Electronic Patient Record. The tool interacts with the physician via a user-friendly graphical interface as well. In particular, GLARE also adopts advanced simulation, temporal-reasoning and decision-theory techniques in order to assist physicians in their decision-making activities [16,26].

A distinguishing feature of GLARE's architecture is the introduction of an intermediate XML layer between the acquisition/execution (interface) modules, and the databases (where GLs and patients' data are physically stored). XML acts as an interlingua between the highest layer and the DBMS layer: the acquisition and execution modules actually interact only with the XML layer, through which they obtain the knowledge stored into the DMBS. The use of XML as an interlingua allows us to express GLs in a format with characteristics of legibility, and to publish them on the Web, making easier their dissemination. On the other hand, the DBMS layer grants a homogeneous management of the data, by integrating the GL representation with the pre-existent hospital information system in the same physical DBMS. GLARE architecture is depicted in Fig. 3 (where the parts in the dashed box represent the extensions needed to implement our model checking-based approach, and will be described in Section 5).

3.2. GLARE representation language

In GLARE, a clinical GL can be represented as a hierarchical graph, where nodes are the *actions* to be executed, and arcs are the *control relations* linking them.

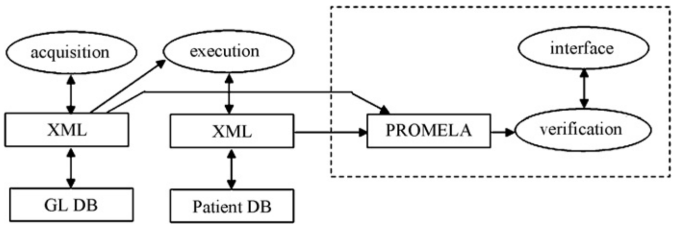


Figure 3. Architecture of the GLARE system: the dashed box includes the verification part, oval nodes represent computation modules, and rectangles represent data.

In GLARE, we distinguish between *atomic* and *composite* actions (plans), where atomic actions represent simple steps in a GL, and plans represent actions which can be defined in terms of their components via the *has-part* relation.

Four different types of atomic actions can be identified:

- *work actions* (depicted as circles in Fig. 4), i.e. actions that describe a procedure which must be executed at a given point of the GL;
- *decision actions* (depicted as diamonds in Fig. 4), used to model the selection among different alternatives. We have introduced a distinction between:
  - *diagnostic decisions*, used to make explicit the identification of the disease the patient is suffering from, among a set of possible diseases, compatible with her findings. A diagnostic decision is represented as an open set of triples ⟨diagnosis, parameter, score⟩ (where, in turn, a parameter is a triple ⟨data, attribute, value⟩), plus a threshold to be compared with the different diagnoses' scores;
  - *therapeutic decisions*, used to represent the choice between paths in a GL, where each path represents a particular therapeutic process. The choice can be made by evaluating a fixed set of parameters (effectiveness, cost, side effects, compliance and duration);
- *query actions* (depicted as parallelograms in Fig. 4), i.e. requests of information (typically patient's parameters), that can be obtained from the outside world (physicians, databases, patient's visits or interviews). The GL execution cannot go on until this information has been obtained;
- *conclusions* (depicted as triangles in Fig. 4), which explicitly identify the output of a decision action.

Actions in a GL are connected through control relations. Control relations establish which actions can be executed next, and in what order. In particular, the *sequence* relation explicitly establishes what is the following action to be executed; the *alternative* relation describes which alternative paths stem from a decision action, and the *repetition* relation, states that an action has to be repeated several times. In detail, given a repeated action, the number of its repetitions can be fixed a priori, or, alternatively, it can be asserted that the action must be repeated until a certain *exit condition* becomes true (in this case, the number of repetitions is only known at runtime, during execution). In particular, advanced AI techniques are required to deal with temporal reasoning in repeated actions.

3.3. A running example: the ischemic stroke guideline

As an example, in Fig. 4, we present the flow-chart for the swallowing test contained in the ischemic stroke GL, visualized through the GLARE graphical interface. This guideline about ischemic stroke was made in the ASO S. Giovanni Battista of Turin (one of the largest hospitals in Italy) by a multi-disciplinary group

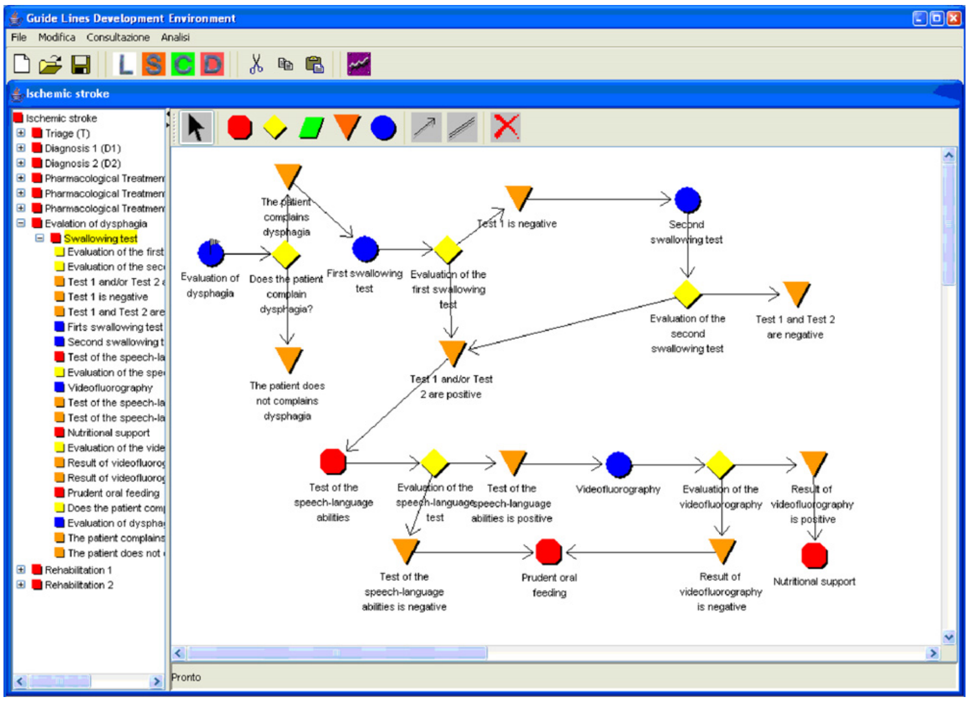


Figure 4. The swallowing test of the ischemic stroke GL visualized through the GLARE graphical interface.

(33 physicians) on the basis of the Italian guidelines for stroke prevention and management (http://www.spread.it/SpreadEng/EnglHome.htm URL last accessed on 22/05/2009). The final version (edited in 2002) is 82 pages long, and was used in the Ministerial project TRiPSS-II ("Sperimentazione di strumenti per l'implementazione di linee-guida"), coordinated by the CeVEAS (Modena) with the participation of the ASL 12 (Biella), ASL 4 (Turin), Mauriziano Hospital (Turin), San Giovanni Battista Hospital (Turin). The paper guideline has been acquired in a computer format using GLARE by a multi-disciplinary team consisting of both expert physicians and knowledge engineers. The computerized guideline focuses on the skeleton procedures, and consists of 319 nodes (actions).

In particular, in the ischemic stroke GL, if the patient complains dysphasia, s/he is submitted to a first swallowing test. If this is negative, a second test is performed and evaluated. If one of the tests if positive, the patient is submitted to the evaluation of a speech-language pathologist. If this evaluation confirms abnormalities, videofluorography is performed and evaluated. In case abnormalities are confirmed, nutritional support is applied (parenteral and/or enteral-tube nutrition). If the evaluation of the speech-language pathologist or the result of videofluorography is negative, prudent oral feeding and re-evaluation are performed.

The ischemic stroke GL will be referred to as a running example in the rest of the paper.

4. SPIN and LTL: a model checking approach to verification

In this section, we introduce model checking approaches to verification and, more precisely, we shortly describe the LTL model-checker SPIN [25] that has been used in the verification of GLs in GLARE.

4.1. Model checking: the general principles

In the model checking approach [18], given a model describing all the possible evolutions of a system and a specification expressed in a temporal logic [27], the model is checked to see whether it satisfies the specification.

The model is usually given in a special-purpose language, which depends on the model-checker, by defining a finite state machine. A finite state machine is, in essence, a directed graph, whose nodes (vertices) represent the states of the system, and whose edges represent the transitions from state to state. Transitions typically correspond to the execution of actions changing the state, or to the occurrence of external events changing the state. As we will see in the following, in our application of model checking to the verification of GLs, the model of the system is built from the GL as well as from the agents interacting with it, and it describes all the possible evolutions of the state of the world in accordance with the GL requirements. Each state of the system defines the patient's parameters (such as laboratory test values, and symptoms), and the state transitions correspond to the executions of actions changing the state of the world (work actions, query actions, etc.—see Section 3.2).

Given the model of the system, the possible executions (*runs*) of the system can be obtained by considering, for each state, all the possible actions which can be executed in that state and in all the resulting successor states. All the system runs that can be obtained from a given initial state can be represented as a tree, as shown in Fig. 5. We have labeled each state with the propositions true at that state, and each transition with the action causing the transition. For instance, in state s1 (in which p and q are true and r is false), two actions can be executed: action a1, which makes p false, leading to state s2, and action a2, which makes r true, leading to state s3. In general, the system runs are infinite sequences of states (and transitions), although here we will be mainly concerned with finite executions, that is, with finite prefixes of runs. An example of a run in Fig. 5 is the infinite sequence s1 s2 s4 s4 s4 . . ., where in the figure we have omitted to represent the infinite repetition of occurrences of the state s4. Similarly, we assume that all the other runs in Fig. 5 are infinite, by regarding the leaf states in the tree to be repeated infinitely many times.

Observe that, in general, the initial state is not unique, as a parameter (for example, *fever temperature*) may be undetermined in the initial situation and this can require keeping into consideration all the different possible values for that parameter,

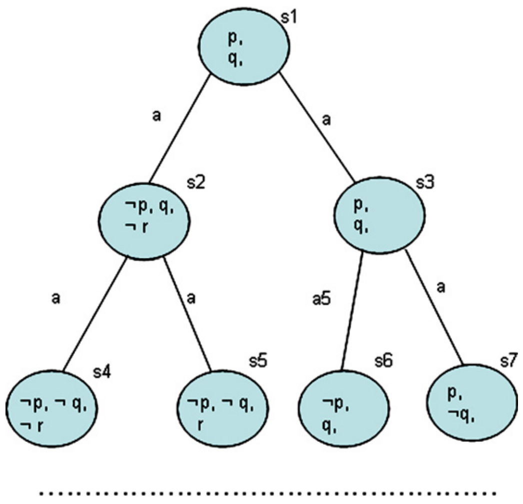


Figure 5. Executions of the system from the initial state s1 = {p,q,¬r}.

thus considering a different initial state for each value (for instance, *high-fever*, *low-fever*, *normal-fever* etc.).

Once a model of the system (i.e., in our application, of the GL and of its executing environment) has been defined, properties of the system itself can be verified, by reasoning on its possible runs. Such properties are *specifications* that the system is expected to satisfy. Here, we are interested in verifying that the GL satisfies desired properties and it does not satisfy undesired ones. Before discussing the matter of verification in further detail, let us shortly describe the model-checker SPIN.

4.2. Model checking in SPIN

Our model checking approach to the verification of GLs makes use of the Linear time Temporal Logic (LTL) model-checker SPIN [25]. In SPIN the specification of the model is given in the input language Promela (that will be shortly described below), and the specification to be checked is a formula of LTL.

Promela allows a high level model of a distributed system to be defined by modelling each process in an extended pseudo C code, including synchronization primitives and message exchange primitives. Promela provides the usual *if-then-else* and iteration constructs of imperative languages, but it also allows for goto statement (allowing jumps to labels), for the non-deterministic choice construct, as well as for the parallel execution of processes. Processes may share global variables and they also may exchange messages through asynchronous communication channels.

SPIN translates each Promela process into a finite automaton, and the global behaviour of the system is obtained by computing an asynchronous interleaving product of automata. The resulting automaton represents the global state space of the system (the model containing all the possible executions – runs – of the GL) and can be built on-the-fly during the verification process.

The correctness claims, that have to be checked on the model of the system, are specified as temporal logic formulas in LTL.

In Section 4.3, we shortly describe how LTL formulas are defined, and how they are checked in a model. Readers who are familiar with LTL can completely skip the rest of this section.

4.3. LTL formulas

LTL formulas are built from the usual connectives AND (&&), OR (||), NOT (!), implication (-⟩) and bi-implication (⟨-⟩) of classical propositional logic, as well as from the temporal operators always ([]), eventually (⟨⟩) and strong until (U). In the following, for sake of readability, we will use the standard notation $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \Box, \Diamond$, U for the connectives, rather than SPIN notation given above.

An LTL formula is evaluated in a linear model which is defined as a pair (σ,V), where σ is a sequence of *worlds* (each world representing a *state* of the system), and V is a valuation function assigning to each world s in σ a propositional valuation V(s), i.e., assigning a (true or false) value to each proposition in each world. An example of a linear model is given by the run σ = s1 s2 s4 . . . in Fig. 5, where the valuation function is given by the propositions labeling each world.

A temporal formula is evaluated at a world s of a model (σ,V). The valuation of the classical connectives is defined as usual, while the valuation of the temporal connectives is defined in the following way. Let σ be the sequence s0 s1 s2 . . .:

(i) A formula $\Box\alpha$ is true in a world sk of σ if α is true in all the worlds sh of σ, for all $h > k$.
(ii) A formula $\Diamond\alpha$ is true in a world sk of σ if there is a world sh of σ, with $h > k$, such that α is true in sh.
(iii) A formula β U α is true in a world sk of σ if there is a world sh of σ, with $h > k$, such that α is true in sh, and β is true in all the worlds si, with $k \le i < h$.

Intuitively, given a model (σ,V), the formula $\Box\alpha$ is true in a world *sk* when α is true in all the worlds following *sk* in σ. The formula $\Diamond\alpha$ is true in a world *sk* when there is a world *sh* reachable from *sk* in which α is true. The formula β U α is true in a world *sk* when there is a world *sh* reachable from *sk* in which α is true, and β is true in all the worlds of σ from *sk* to *sh*. For simplicity of exposition, in the following, we will omit examples involving the *until* operator.

Given a property (specification) as an LTL formula, SPIN verifies if the property is true on all the executions of the system. Namely, each run of the system is regarded as a linear temporal model and the truth of the property is verified on it, from the initial state.

Let us consider, for instance, the run σ1 = s1 s2 s4 . . . in Fig. 5. The formula $\Box\neg$r is true on the run σ1 in the initial world s1. In fact, ¬r is true in all the worlds reachable from s1, namely s2 and s4. If we consider run σ2 = s1 s2 s5 . . ., $\Box\neg$r is not true on this run in s1, as there is a reachable state s5 in which ¬r is false. The formula $\Diamond\neg$q is also true on the run σ1 in the initial world s1, as there is a state s4 reachable from s1 in which q is false (¬q is true). Instead, the formula $\Diamond\neg$q is false on the run σ3 = s1 s3 s6 . . ., as there is no reachable state from s1 on which q is false.

The formula

$$\Box(\neg p \rightarrow (\neg q \vee \neg r)), \qquad (1)$$

(meaning that, in all the reachable states, if p is false, then either q or r is false) is true on run σ1 (in the initial state s1). Such a formula is also true on the runs σ2 = s1 s2 s5 . . . and σ4 = s1 s3 s7 . . ., while it is not true on σ3 = s1 s3 s6 . . ., as in state s6 p is false, but neither q nor r is false. Hence, if SPIN were asked to check whether formula (1) is true on all the runs of the system, the answer would be NO and run σ3 would be returned as a counter-example to the property (see below).

The property

$$\Box(\neg q \wedge \neg r \rightarrow \neg p)), \qquad (2)$$

instead, is true on all the four runs, and SPIN would answers YES to the request of verifying formula (2) in all possible runs.

The property

$$\Diamond(\neg q \wedge \neg p), \qquad (3)$$

(eventually a state can be reached in which both p and q are false), is true on runs σ1 and σ2 but not on σ3 and σ4. Instead, the property $\Diamond$ (¬q∨¬p) is true on all the runs.

Consider the system has a further run $\sigma5$ (not represented in Fig. 5), with $\sigma5$ = s8, s3, s3,. . .., where s8 = {p, ¬q, ¬r}. Property $\Diamond(\neg q \vee \neg p)$ is not true on $\sigma5$. However, property

$$p \wedge q \rightarrow \Diamond(\neg q \vee \neg p), \tag{4}$$

(if p and q hold in the initial state, then eventually either q or p will be false) holds on all the five runs (including run $\sigma5$, on which (4) holds trivially, as the antecedent of the implication ($p\wedge q$) is false in its initial state s8).

In general the temporal modalities can be arbitrarily nested within formulas. For instance, we can write a property

$$\Box(p \wedge q \rightarrow \Diamond(\neg q \vee \neg p)), \tag{5}$$

meaning that, for all the reachable states, if $p\wedge q$ is true, then $\neg q\vee\neg p$ will eventually be true. Although this property holds on all the runs in Fig. 5, it does not hold on run $\sigma5$. In fact, there is a reachable state s3 of $\sigma5$ in which $p\wedge q$ is true, but from s3 there is no reachable state in which $\neg q\vee\neg p$ holds.

To verify that a formula holds on all the runs of the system, SPIN converts the negation of the temporal formula into a Büchi automaton and computes its synchronous product with the system global state space. If the language of the resulting Büchi automaton is empty then the property is true on all the possible execution of the system, otherwise the verifier provides a counter-example for the property (an execution path on which the property is false).

Besides verifying that a formula holds on all the runs of the system, SPIN also allows to check if there is a run of the system on which a given formula α is true. This is done by negating the formula α, and then verifying whether the formula $\neg\alpha$ holds on all the runs of the system. If not, the resulting counter-example for $\neg\alpha$ is a run on which α is true. In the following, we will be interested in both kinds of verifications: the verification that a property holds on all the runs, and the verification that there is a run satisfying the property.

5. Integrating GLARE with SPIN

As a proof of concept, we have applied the general methodology described in Section 2 in order to couple GLARE with the model-checker SPIN.

As shown in Fig. 3 and explained in Section 3.1, in GLARE we provide a three-level representation of GLs: in an internal system format (i.e. in the GLARE representation language, at the acquisition and execution modules level), in XML, and in the DBMS. Given its readability and its generality, we have chosen the XML-format representation as the basis of the translation. In other words, our translator takes in input a GLARE GL, expressed in the XML format, and transforms it into the corresponding GL in the Promela language.

Analogously, the patient data (taken from the XML format) are also translated into the Promela language.

In this section, we briefly describe the general principles we adopt to convert a GL in the GLARE formalism into the corresponding agent-based program in the Promela language. First, we describe how a GL in the GLARE formalism is mapped to a set of interacting processes (called *agents* henceforth), i.e. to a set of Promela processes and to a set of proper synchronization primitives and message exchange primitives, and in particular we show the pseudo-code corresponding to a query action. Then, we shortly describe our translator module.

5.1. Guidelines as agents

Obviously, the basic object we need to represent in the Promela formalism for the purpose of verification is the GL itself. As already observed, a GL can be seen as a set of actions, to be executed in the order specified by a set of control flow primitives. As detailed below, we have mapped each construct (action or control flow primitive) in the GL (see Section 3.2) to a Promela statement or to a Promela piece of code.

However, GL execution is a complex phenomenon that cannot be modelled just by representing the GL *per se*.

In the following, we propose a possible, more realistic way of capturing the dynamics of the GL and of its execution environment, based on the idea of modelling a set of processes, whose interaction models the GL execution itself. Actually, the identification of the involved processes and of the characteristics of their interaction represents an additional, original contribution of this paper, which is mostly independent of GLARE's approach.

One of the required processes – which we will call *agents* – is of course the GL itself. The other agents represent the (human or not) components interacting with the GL at execution time.

In particular, the *Database* agent has to be represented. Actually, patient's characteristics need to be specified, and, rather naturally, we characterize a patient by relying on her data, which are typically maintained in the clinical *database*. The Database agent thus provides data on demand, and is able to store new data values.

Updated data values are sometimes obtained from additional sources (e.g. from the hospital laboratory service). We have generically modelled such sources and services by means of a further agent, called *Outside* world.

Last but not the least, GL execution is performed by a *physician*; therefore, the physician's behavior needs to be modelled as an agent as well. In particular, we have identified two main tasks that the Physician agent is expected to cover when applying a GL to a specific patient. Obviously, it is required to make decisions, i.e. it has to select exactly one diagnosis or therapy, among a set of alternative ones. Moreover, it has to evaluate data recency and reliability: if a data value, extracted from the database, is judged as unreliable or not up-to-date (i.e. too old), the Physician agent has to rise the problem, thus triggering the generation of a newer data value from the outside world.

In summary, the model of the distributed system we propose to simulate GL execution can be described by the interaction among the following agents, interpreted as Promela processes:

(i) the Guideline agent, which models the overall behavior of the GL;
(ii) the Database agent, which models the behavior of the patient database, allowing for data insertion and retrieval;
(iii) the Outside agent, which represents the outside world and provides up to date values for patient data (together with the time of their measurement) when they are not already available in the database or are evaluated as being not reliable by the physician. It also stores data in the database, and simulates the execution of work actions by reporting their success or failure;
(iv) the Physician agent, which interacts with the GL by evaluating the patient data, choosing among the different alternative feasible paths as a physician would do, and judging data reliability. Observe that we model the Physician agent as a non-deterministic process, since it is not possible to know a priori all the possible choices of physicians in all the possible situations. We therefore model the uncertainty about the choice of physicians using non-determinism: from the point of view of the simulation, choices are taken randomly by the Physician agent.

Details about each agent's representation are provided below.

Observe that such a general interpretation and representation of the GL execution environment as a distributed system does not depend on the adoption of GLARE and of its representation language, but is very general indeed, and is mostly adaptable to other approaches to GL modelling and verification. Also note that such an interpretation, if properly extended, would be even more useful to describe the execution of complex hospital workflow processes, which heavily rely on the communication among interacting agents: workflow processes will be the object of our research in the future.

(i) *Guideline agent*: In order to model the Guideline agent, we had to map each construct in the GL (see Section 3.2) to a Promela statement or to a Promela piece of code. As regards control relations, in particular, sequence is mapped using the goto statement towards the next action in the GL. Observe that each action has a label (Et in the example below) that identifies its address; this label is used as the action name and as the target of goto statements. The alternative relation is also mapped using the instruction goto, but in this case its target (i.e. the next action to be executed) is not determined in a static way, but is chosen on the basis of Physician agent's answer. The repetition relation is modelled by the repetition construct (do statement) in Promela.

As regards GL atomic actions, let us first take into account our representation of query actions in Promela pseudo-code:

```
 Et:
        cost_of_done= 0;
   for each required datum {
                LGtoDB!datum.D,datum.A;
                LGfromDB?datum.D,datum.A,datum.V,datum.T,datum.C;
   if (datum.V == MISSING) then        {
        LGtoOUTSIDE!datum.D,datum.A
        LGfromOUTSIDE?datum.D,datum.A,datum.V,datum.T,datum.C;
        cost_of_done = cost_of_done + datum.C;
        }
   else {
        LGtoPH_data!datum.D,datum.A,datum.V,datum.T;
        LGfromPH_data?datum.D,datum.A,valid;
     if NOT(valid) then
        {
        LGtoOUTSIDE_data!datum.D,datum.A;
   LGfromOUTSIDE_data?datum.D,datum.A,datum.V,datum.T;
        cost_of_done = cost_of_done + datum.C;
        }
        fi;
        }
   fi;
   }
   done = Et
 goto NextEt
```

In the pseudo-code above, for each datum, the Guideline agent sends a request message to the Database agent (i.e. LGtoDB! datum.D, datum.A) and waits for the answer message (LGfromDB? datum.D,datum.A, datum.V, datum.T). Observe that the datum is a quadruple ⟨ D, A, V, T⟩, where D is the category to which it belongs (e.g. liver examination), A is the attribute of interest (e.g. volume of the liver), V is the value assumed by the attribute in the given case, and T is the time at which the measurement was taken. Then it checks whether the datum is missing (datum.V == MISSING); if the datum is not stored in the patient database, it sends a request message to the Outside agent (LGtoOUTSIDE! datum.D, datum.A) in order to obtain such a value, and waits for its answer message (LGfromOUTSIDE? datum.D, datum.A, datum.V, datum.T). Otherwise if the datum is found, it sends a request message to ask the Physician agent to evaluate whether the datum is still reliable (LGtoPH_data! datum.D,datum.A,datum.V,datum.T;), or it is too old. Notice that, in GLARE, every datum has an associated timestamp (e.g., datum.T). It then waits for the answer of the physician (LGfromPH_data? datum.D,datum.A,valid). If the datum is not valid, the Guideline agent sends a request message to the Outside agent in order to provide up to date values for this datum.

Notice that the above translation provides as an output, among the other things, a set of variables (one variable for each requested datum), which will be used in order to store the values of the requested data in the GL. These variables can be further on referred to in the LTL properties to be verified.

The variable cost_of_done, on the other hand, stores the cost of the GL actions and is updated when a new value of the datum is acquired (in case the datum is missing and/or is not reliable). The variable done records the last executed action and is updated when a new action is performed with the label of such an action.

The statement goto NextEt is used to jump to the next statement. As mentioned above, GLARE sequence construct is translated by making use of goto statements.

In the following, we provide a short and qualitative description of the translation of the other types of actions in GLARE.

A decision action is translated into a Promela piece of code in which, for each alternative, the Guideline agent evaluates the support conditions on the basis of patient's parameters and sends the support values to the Physician agent; then, it waits for the Physician agent to send the answer message that points out the next action to be performed.

A work action is managed similarly. First of all, it is checked whether the required resources are available. If not, the action fails, otherwise the action can be executed and the process modelling the work action asks the Outside agent to "execute" the action and waits for its answer. If the Outside agent answers that the action has been "executed" correctly, the execution of Guideline agent continues as follows: in case of acyclic work actions, there is a jump to the next action in the guideline; in case of cyclic actions, a test evaluates whether a new iteration has to be executed. If a failure is reported (e.g. a resource is not available or the Outside agent answers that the action has failed for other reasons), the Physician agent is asked to choose among the available "backtracking" points, i.e., points in the GL from which the execution can continue.

A conclusion action simply consists in a Promela piece of code that prints a message, since it represents the explicit output of a decision process.

Finally, a plan consists of a Promela piece of code that checks whether the required resources (preconditions) are available. If so, it jumps to the first action of the plan. Otherwise the plan fails. Observe that the actions in the plan are executed according to the order defined in the GL and that, after the execution of the last action of the plan, there is a jump to the label of the next action in the guideline, in case the plan is acyclic, or a jump to a statement which checks whether a new iteration of the plan is needed, in case the plan is cyclic.

In the following, we describe the other four agents (for the sake of brevity, the Promela code is not shown).

(ii) *Database agent*: The Database agent models the behavior of the patient database. It waits for a message of data retrieval/ insertion. If it receives a request message asking to retrieve a clinical datum for a specific patient, it sends the value to the Guideline agent. Observe that if the datum is not stored in the patient's database, the Database agent answers that the datum is missing (i.e. datum.V is set to the value MISSING). If it receives a request message asking to insert a value for a datum, it performs the insertion of this information into the clinical database.

(iii) *Outside agent*: The Outside agent waits for a message from the Guideline agent. It can receive two types of requests: provide an up to date datum or execute a work action. If it receives a request message asking to provide up to date values for a datum, it provides in a non-deterministic way the new value (i.e., it randomly chooses a value between the possible ones), asks the Database agent the insertion of this datum, and reports this datum to the Guideline agent. If it receives a request message asking to execute a work action, it performs this action and reports about its success or failure to the Guideline agent.

(iv) *Physician agent*: The Physician agent waits for a message from the Guideline agent. As already observed, it provides the requested information in a non-deterministic way (i.e. its choice is a random one) and sends a message containing this information to the Guideline agent.

Note that Database agent, Outside agent, and Physician agent spend their time waiting for a request from another agent, and then processing such a request.

Let us point out that the above mapping to a Promela specification has been adopted considering only qualitative constraints on the temporal ordering of actions (instead of more complex temporal constraints, such as the fact than an action must be performed during another one, or must finish at least 1 h before the end of another one; see [16] for more details about temporal constraints in GLARE). More precisely, in our Promela specifications, the actions can be executed either sequentially or concurrently, and no additional temporal constraint between actions is considered.

The timestamps associated with data in GLARE (describing the time at which the measurement of each datum is taken) are not required for evaluation of constraints in the Promela specification. In fact, they are only used by the Physician who has to decide if the data are still reliable or not. As we model the Physician as a non-deterministic process answering the requests from the guideline, we do not need to have time values available during the computation and hence to represent them in the model: the Physician non-deterministically recognizes a datum as being valid or not. Therefore, we only need to represent the fact that the timestamps are exchanged between the guidelines and other agents while we do not need an explicit representation of time values. This simplification, which is made possible by the fact that GLARE's temporal constraints between actions are not considered, highly reduces the complexity of the model.

5.2. The translator

As explained above, we have defined a translator which takes a set of XML documents representing any GLARE GL and automatically transforms them into the corresponding GL in the language Promela.

A GL in GLARE is a hierarchical graph, in which it is possible to have composite actions (i.e. plans), which can be defined in terms of their components via the *has-part* relation (see Section 3.2). In the XML document such a structure is maintained. Thus, the translator works as a top-down parser, according to the pseudo-code below:

```
translator(graph <N,E>, vocabulary V)
{
x ← empty
preprocessing(<N,E>, V, x)
generateAgentsCode(<N,E>, V, x)
}
```

In particular, the translator takes in input a graph defined as a couple ⟨N,E⟩ (where N is the set of nodes and E is the set of edges), which is the XML document representing the GL, and the vocabulary V, which contains the medical data information. To make the translation, the parser visits the graph twice. The first time it makes a preprocessing (i.e. it calls the `preprocessing` function) in order to obtain the data concerning the requests of information of query actions, according to the pseudo-code below (i.e. such data are stored in *x*):

```
preprocessing(graph <N,E>, vocabulary V, list x)
{
for each node n in N{
    if (n is a Query Node)
      for each parameter requested in n, add to x the pair <parameter, list of the possible
values> taking the possible values of the parameter from V
    if (n is a Plan Node)
      preprocessing(<N',E'>, V, x) where <N',E'> is the subgraph
that defines n
}
```

The `preprocessing` function visits in-depth the graph G. This preliminary step is needed due to the fact that the model-checker needs that all possible variable values are known. Indeed, such values are defined in the vocabulary V. Therefore, for each parameter (e.g., fever), the list of its possible values is collected from the medical vocabulary (e.g., absent, low, medium, high). Observe that in ⟨*parameter, list of the possible values*⟩ the second element *list of the possible values* is always finite, since continuous values are always discretized in the vocabulary.

In the second step, the parser visits the graph for the second time, through the function `generateAgentsCode`, in order to build the agents shown in Section 5.1, which model the GL behavior, according to the pseudo-code below:

```
generateAgentsCode(graph G, vocabulary V, list x)
{
for each node n in N{
        if (n is a Query Node)
                generate Promela Code for: Guideline agent, Physician
                agent, Database agent, Outside agent
        if (n is a Decision Node)
                generate Promela Code for: Guideline agent, Physician
                agent, Database agent
        if (n is a Action Node)
                generate Promela Code for: Guideline agent, Database
                agent, Outside agent
        if (n is a Conclusion Node)
                generate Promela Code for: Guideline agent
        if (n is a Plan Node)
                generate Promela Code for: Guideline agent, Database
                agent, Outside agent
                generateAgentsCode(<N',E'>, V, x) where <N',E'> is
the subgraph that describes n
}
for each edge e in E{
        generate Promela Code (goto labels) concerning edges for
        Guideline agent
        }
}
```

During this second step, the parser first visits each node of the graph, and for each node in the set N it generates the Promela code that corresponds to the node's intended behavior. Observe that, since each type of node has a different behavior in the GL, the parser discriminates between the different types in order to generate the correct code. For instance, consider the case of the query action: the parser generates the code for the Guideline agent, which will interact with the Physician agent, the Database agent, and the Outside agent, since all these four agents are involved in implementing the query action execution in practice (see Section 5.1 for the explanation of interaction between these agents) and for the Promela pseudo-code for the Guideline Agent concerning the translation of a query action).

Then, the parser visits each edge of the graph and completes the Guideline agent code. In particular, it creates the code connection in the Guideline agent between the Promela piece of code that describes each action and the piece of code that describes the next action, according to the control flow. Observe that each piece of Promela code corresponding to an action (see Section 5.1) is identified by a label (i.e. Et in the example in Section 5.1) and the code ends with a *goto* statement that allows to reach the next action (i.e. goto *NextEt* in the example in Section 5.1). Thus the parser instantiates the value of the label NextEt.

Observe that, if we want to evaluate some properties on the execution of a GL for a specific patient (i.e. the properties are dependent of the patient's data), the translation process needs to take into account the patient's data. The patient's data, taken from the corresponding XML document, are then introduced into the Promela code. Having a patient data document means that the GL is instantiated on a specific patient and that some clinical data of this patient are known. Thus, the translator takes these data from the XML document and sets the values of the corresponding variables in the Promela document. Of course some data might be missing, or their value could change during the GL execution (e.g. a datum can be evaluated as not being reliable by the Physician agent). In the case we want to evaluate properties that are independent from the patient's data, the patient's data are not needed and they are not given to the translator. In such a case, the value of the variables corresponding to the patient's data is not instantiated, and the model-checker tests the properties on all the possible values of the variables.

As a final remark, notice that the translator allows any GL representation in GLARE to be mapped into the corresponding Promela code. We think that the adoption of a high-level process-based language like Promela makes this translation relatively easy and natural, as shown in the brief description above. We believe that the simplicity of the translator gives a measure that the representation of the guideline in GLARE and in Promela is not so far from each other.

6. GL properties

In the previous sections, we have discussed the advantages of adopting model checking techniques in the verification of GL properties, and we have described a loosely coupled approach to combine the SPIN model-checker with GLARE. In Section 2 we have sketched the different phases in the GL life-cycle, identifying in the meanwhile different classes of properties that can be checked in each phase. Until now, however, we have relied on the readers' intuition as regards what are the properties to be checked. As a matter of fact, temporal logics such as LTL allow one to express a wide range of formulas. Such an expressiveness and generality motivates a deeper analysis of what kinds of properties, expressible in LTL, are useful in the GL context.

We divide the properties on the basis of the GL life-cycle phases we pointed out in Section 2.1. It is worth stressing that, in the paper, we do not focus only on the properties to be checked in the acquisition phase to grant the *consistency* of GLs (as done for instance in Protocure). We also take into consideration properties that can be useful in the other stages of the GL life-cycle, including GL execution on specific patients (e.g., checking the existence of paths which use – or do not use – a given resource, or a given treatment, or perform – or do not perform – a given action). Of course, we do not claim we are exhaustive: we regard such an analysis as a first step to identify ''relevant'' or ''frequently-used'' patterns of properties, with the final goal of providing (in our future work) a user-friendly interface for them. This analysis is, in our opinion, one of the core contributions of this paper, and is presented in the rest of this section.

Although in many examples the reference GL we considered is the ischemic stroke GL (with specific reference to the GL developed by Azienda Ospedaliera S. Giovanni Battista, in Turin, and acquired in GLARE—see Section 3), we also take some examples from other GLs. In order to characterize the expressiveness of LTL with respect to GL property verification, we have distinguished the properties on the basis of the "pattern of operators" they use. In particular, as described in section 4, given a property α (expressed by an LTL formula), we can either check if it holds on all the executions of the guideline, or if there is a run on which it holds. For the sake of exposition, we describe the properties to be verified by distinguishing two components:

(1) a quantifier on "runs": $\forall$, stating that we verify if the property holds on all the runs, and $\exists$, stating that we look for one run satisfying the property
(2) an LTL formula (as described in Section 4.3).

Therefore, in the following, we formulate properties to be checked as pairs

⟨**run_quantifier, LTL_formula**⟩

For the sake of clarity and simplicity, in the first examples below we describe properties that can be expressed by simple LTL formulas containing only one temporal operator (of course, more complex examples can be built using more than one temporal operator and some examples will be shown in Sections 6.5 and 6.6). Each property is prefixed with the combination of ⟨quantifier, temporal_operators⟩, where the temporal operators are those occurring nested in the query, and is followed by the indication (in parenthesis) of the GL it refers to.

6.1. Structural properties

As discussed in Section 2.1, structural properties concern the existence of the appropriate clinical requirements, and are relevant in order to ensure the appropriate management of any patient.

Here and in the following, we assume that the SPIN variable "done" is set, in each state, to the action performed in such a state.[1]

Class: $\forall \, \Box$

Example: Verify that neurological deficit is always present (ischemic stroke GL)

Property 6.1.1: ⟨$\forall$run, $\Box$(neurological deficit = present)⟩

Comment: The property is true if neurological deficit is always present, in all the states of all possible runs.

Relevance: The ischemic stroke GL is not applicable to a patient with no neurological deficit.

Class: $\exists \, \Box$

Example: Verify that the GL contains a run not including any surgical intervention (cholelithiasis GL)

Property 6.1.2: ⟨$\exists$run, $\Box$(done $\neq$ surgery)⟩

Comment: The property evaluates to true if there is at least a run in which surgery is never performed.

Relevance: The most frequent treatment of gallstones is the expectant management (asymptomatic gallstones).

Class: $\forall \, \Diamond$

Example: Verify that any run contains antibiotic treatment (community acquired pneumonia GL)

Property 6.1.3: ⟨$\forall$run, $\Diamond$ (done = antibiotic_treatment)⟩

Comment: The property evaluates to true if all possible runs contain a state in which an antibiotic treatment is administered.

Relevance: The antibiotic treatment is mandatory in the case of community acquired pneumonia.

Class: $\exists \, \Diamond$

Example: Verify that the GL contains a run including thrombolysis (ischemic stroke GL)

Property 6.1.4: ⟨$\exists$run, $\Diamond$(done = thrombolysis)⟩

Comment: The property evaluates to true if there is at least a run in which thrombolysis is executed.

Relevance: It is important to perform a thrombolysis in case the patient is eligible and in the hospital is present a stroke unit.

In the following, we mention some more complex formulas, involving some nesting of temporal operators.

Class: $\forall \, \Box\Box$

Example: Verify that cholecystectomy is not repeated (cholelithiasis GL)

Property 6.1.5: ⟨$\forall$run, $\Box$ (done = cholecystectomy $\rightarrow$ $\Box$ (done $\neq$ cholecystectomy))⟩

Comment: Once removed, an organ cannot be removed again.

Relevance: The surgical treatment of gallstones in a cholecystectomized patient cannot consist of a new cholecystectomy.

6.2. Medical validity properties

As discussed in Section 2.1, medical validity properties concern both the exclusion of dangerous treatments and the inclusion of the most appropriate treatments for the considered class of patients, and are relevant in order to ensure best practice. In the following examples, predicates must be introduced in order to model the patients' data. Consistently with the formalism used in the clinical records in GLARE (as well as in several hospital information systems), we represent patient clinical data as triples ⟨data, attribute, value⟩. Specifically, in the following temporal properties, we adopt the notation: "data_attribute = value". Notice, however, that the results in this paper are completely independent of such a notational choice.

Class: $\forall \, \Box$

Example: Verify that whenever hepatic encephalopathy is present, diuretics are not administered (as cites GL)

Property 6.2.1: ⟨$\forall$run, liver_state = encephalopathy $\rightarrow$ $\Box$(done $\neq$ diuretics_administration)⟩

Comment: Diuretics are contraindicated in hepatic encephalopathy.

Relevance: Diuretics can worsen the liver perfusion and precipitate the encephalopathy or worsen its severity.

Class: $\exists \, \Box$

Example: Verify that there is a run in which the acetylsalicilic acid (ASA) is not used (ischemic stroke GL)

Property 6.2.2: ⟨$\exists$run, $\Box$done $\neq$ ASA⟩

Comment: This is the condition for applying the GL in the case the patient is allergic to ASA.

Relevance: The ASA allergy is potentially life threatening.

Class: $\forall \, \Diamond$

[1] It is worth noting that, in the formulation of the properties in this section, for the sake of clarity we have assumed that properties are checked considering the reference GL being specified. Therefore, we do not explicitly model the context (GL) in which the properties are verified. For instance, the third property implicitly assumes that 'community acquired pneumonia' holds. Removing the above assumption is trivial: the property could be more exhaustively expressed as: ⟨$\forall$run, community_acquired_pneumonia = yes $\rightarrow$ ($\Diamond$ (done = antibiotic_treatment))⟩.

Example: Verify that if vital signs are altered, the patient is sent to intensive care unit (ischemic stroke GL)

Property 6.2.3: $\langle \forall$run, vital_signs_value = altered $\rightarrow \Diamond$ (done = sent_to_intensive_care_unit)$\rangle$

Comment: Monitoring and treatment in an intensive care unit is preferable in this case.

Relevance: The admission of the patient with altered vital signs in an intensive care unit is mandatory.

Class: $\exists \Diamond$

Example: Verify that, in the presence of ureteral lithiasis, there is at least one state in which endoscopic removal is considered (urinary stones LG)

Property 6.2.4: $\langle \exists$run, ureteral_lithiasis _state = present $\rightarrow \Diamond$ (done = endoscopic_removal)$\rangle$

Comment: Alternative treatments (e.g., surgery) are possible, but more invasive.

Relevance: The endoscopic removal of urinary stones, whenever possible, is preferable.

6.3. Contextualization properties

As discussed in Section 2.1, contextualization properties lead from (general) GLs to hospital-specific GLs, mainly as concerns the presence/absence of instrumentation. These properties are relevant to adapt the general guidelines to the specific environment (hospital).

Class: $\forall \Box$

Example: Verify that each action costs less than x (any GL)

Property 6.3.1: $\langle \forall$run, $\Box$cost_of_done $\langle x \rangle$

Comment: In this example, we assume that, in each state, the SPIN variable cost_of_done represents the maximum allowed cost of the action performed.

Relevance: Cost reduction is one of the most important aspects of health care policy.

Class: $\exists \Box$

Example: Verify that there is a run in which the CT scanner is not used (ischemic stroke GL)

Property 6.3.2: $\langle \exists$run, $\Box$done $\neq$ TC$\rangle$

Comment: If this condition holds, the GL (or, at least a part of it) can be applied also in hospitals where the case the CT scanner is not available.

Relevance: The CT scanner is very important in some cases but not always accessible.

Class: $\forall \Diamond$

Example: Verify that if the CT scanner is not available, patient transfer to another hospital is considered to perform the test (ischemic stroke GL)

Property 6.3.3: $\langle \forall$run, CT_scanner = absent $\rightarrow \Diamond$done = patient_transfer$\rangle$

Comment: The suspect of ischemic stroke should always be confirmed by CT.

Relevance: The CT scan is almost always diagnostic of the ischemic event and influences the subsequent decisions.

Class: $\exists \Diamond$

Example: Verify whether MR is used in the GL (ischemic stroke GL)

Property 6.3.4: $\langle \exists$run, $\Diamond$done = MR_imaging$\rangle$

Comment: This property should be verified since, in the case MR imaging is considered, its execution must be possible.

Relevance: The MR imaging, when available, is the most diagnostic test in some cases of ischemic stroke.

6.4. Properties about the application of a GL to a specific patient

As discussed in Section 2.1, these properties are relevant to ensure a patient-centred approach, considering the peculiarity of each patient.

The formulas in this subsection are quite close to the ones in Section 6.2 since, in both cases, the status of patient needs to be modelled. However, in Section 6.2 the formulas regard "classes" of patients, and are aimed to check the eligibility of the given GL to treat such classes, while here we consider the applicability of the given GL to a specific patient. As a main consequence, while the formulas in Section 6.2 refer to conditions on the status of the available resources in the initial state, here we take into account the clinical conditions of a single patient (fever, pain, infections, etc), which may change during the guideline execution, and, in general, we are interested in the values at different times. For instance, in the property 6.4.1, the first temporal operator in "$\Box$(abdominal_pain_value = acute $\rightarrow \ldots$)" is used in order to model that we are interested not only in the first state of the execution of the GL, but also in all the states in which, during the execution, acute abdominal pain arises. This is the main reason for which properties in this section are "prefixed" with an additional $\Box$ temporal operator.

Of course, the properties below might also be checked considering the specific status of the execution of the given GL on the patient at hand. Technically speaking such an effect can be achieved in our approach by modelling (part of) the past execution in the property to be checked.

Class: $\forall \Box\Box$

Example: Verify that, after a cerebral hemorrhagic event, no anticoagulant drug is administered (ischemic stroke GL)

Property 6.4.1: $\langle \forall$run, $\Box$ (cerebral_hemorrhagic_event = present $\rightarrow \Box$ (done $\neq$ anticoagulant_drug_ administration))$\rangle$

Comment: In the natural evolution of an ischemic stroke, intra-cranial bleeding may be life-threatening

Relevance: The intra-cranial hemorrhage is an absolute contraindication to anticoagulant drug administration.

Class: $\exists \Box\Box$

Example: Verify that, if an infection arises, there is a treatment not based on penicillin

(community acquired pneumonia GL)

Property 6.4.2: $\langle \exists$run, $\Box$(infection_value = present $\rightarrow \Box$(done $\neq$ penicillin_administration))$\rangle$

Comment: Notice that, e.g., penicillin treatment cannot be administered to allergic patients.

Relevance: Antibiotic treatment is based on different drugs which should be adapted to the patient's different conditions.

Class: $\forall\Box \Diamond$

Example: Verify that, if hyperpyrexia appears, hemoculture is performed (FUO GL)

Property 6.4.3: $\langle\forall$run, $\square$(hyperpyrexia_value = present $\rightarrow \diamond$ (done = hemoculture))$\rangle$

Comment: In each run, hemoculture must be performed, if hyperpyrexia appears.

Relevance: Hemoculture is very important in the diagnostic process of the FUO because it can positively influence the antibiotic choice.

Class: $\exists \square \diamond$

Example: Verify that there is a treatment in which growth factors are administered, when leukopenia appears (lymphoma treatment GL)

Property 6.4.4: $\langle\exists$run, $\square$(leukopenia_value = present $\rightarrow \diamond$ (done = growth_factors_administration))$\rangle$

Comment: The growth factors administration can positively reduce the duration of the leukopenia and the risk of infections.

Relevance: If leukopenia is not severe, there are also alternative treatments to the administration of growth factors (e.g., expectant treatment and monitoring). That is why we check the existence of one run in which (in a given status) growth factors are administered, without forcing that they are administered in all runs (in contrast with the verification above).

6.5. "Complex" properties

In the above subsections, for the sake of clarity, we focused our attention on properties that can be expressed quite synthetically, mostly using just one temporal operator (besides the quantifier about runs). Of course, LTL has no restriction on the number of such operators, so that arbitrarily complex formulas can be used to express properties of a GL "per se" or properties about eligibility, contextualization, or application to specific patients (and the logic also allows one to verify properties that are a "mixture" of the above mentioned cases).

In the following, we just propose two additional examples, demonstrating how paths of actions and/or sequences of patient's states can be used in the verification. We refer to the part of the GL about ischemic stroke we described in Section 3.3, Fig. 4.

Class: $\exists\diamond\diamond\diamond$

Example: Verify whether the GL can apply to patients having given features (monitored at subsequent states; ischemic stroke GL)

Property 6.5.1: $\langle\exists$run, $\diamond$(dysphagia_value = present $\wedge \diamond$ (swallowing_test1_value = positive $\wedge \diamond$ (speech_language_evaluation = positive $\wedge \diamond$ (videofluorography_value = positive)))

Comment: This sequential approach is recommended in evaluating dysphagia.

Relevance: A complete evaluation of dysphagia should include all the above aspects sequentially.

Class: $\forall\square\diamond\diamond\diamond\diamond$

Example: Verifying whether the GL always prescribes the above sequence of actions for patients with dysphagia (ischemic stroke GL)

Property 6.5.2: $\langle\forall$run, $\square$(dysphagia_value = present $\rightarrow \diamond$ (done = swallowing_test1 $\wedge$ swallowing_test_value = positive $\rightarrow \diamond$(done = speech_language_test $\wedge$ speech_language_evaluation = positive $\rightarrow \diamond$(done = videofluorography $\wedge$ videofluorography_value = positive $\rightarrow \diamond$(done = artificial_nutrition)))))

Comment: A correct treatment (artificial nutrition) is mandatory in confirmed dysphagia.

Relevance: To perform a correct artificial nutrition all the above diagnostic aspects of dysphagia must be evaluated.

6.6. Incremental property checking

We have mentioned above that, thanks to its flexibility, our approach also supports incremental and interactive verification of GLs. Instead of expressing complex "monolithic" properties such as the ones in Section 6.5, our approach also supports an incremental verification process, in which each part of the GL can be tested For example, in the following we list the sequence of "elementary" checks which can be incrementally and interactively verified.

$\langle\forall$run, $\square$(dysphagia_value = present $\rightarrow \diamond$ (done = swallowing_test1))$\rangle$
$\langle\forall$run, $\square$(swallowing_test_value = positive $\rightarrow \diamond$ (done = speech_language_test))$\rangle$
$\langle\forall$run, $\square$(speech_language_evaluation = positive $\rightarrow \diamond$(done = videofluorography))$\rangle$
$\langle\forall$run, $\square$(videofluorography_value = positive $\rightarrow \diamond$(done = nutritional_support))$\rangle$

In the following example, the result of a test influences the definition of the next test to be performed). Such flexibility is a peculiar advantage of the methodology we propose, and cannot be (at least in general) achieved in conventional approaches, in which the types of verifications need to be specified a priori (since a specific algorithm for each type of specification need to be devised).

A physician may ask whether, given a patient with fever, the ischemic stroke guideline suggest the execution of a blood culture.

$\langle\exists$ run, (fever_value = present $\rightarrow \diamond$ (done = blood_culture))$\rangle$

If the result of the check is positive, then the physician may further check whether, in the above context, the guideline suggests a culture-based antibiotic treatment.

$\langle\exists$ run, (fever_value = present $\rightarrow \diamond$ (done = culture-based_antibiotic_treatment))$\rangle$

Otherwise, if the result of the check about blood culture is negative, the physician may check whether the guideline suggests an empirical antibiotic therapy to cope with fever.

$\langle\exists$ run, (fever_value = present $\rightarrow \diamond$ (done = empirical_antibiotic_therapy))$\rangle$

7. Verifying the ischemic stroke guideline

In this section, we exemplify the adoption of our approach to GL verification, considering the GL about ischemic stroke sketched in Section 3.3.

As discussed in Section 3.3, such a GL was issued by the hospital S. Giovanni Battista of Turin as an adaptation of a national GL, and was later acquired using GLARE. During the *acquisition phase*, several structural and medical validity properties have been checked using our approach. A few of such checks have allowed us to discover inconsistencies in the GL. The verification of one of such checks is reported in the following, as an example.

Example 1(a).

"if an anticoagulant treatment has been excluded at some point of the guideline, later on the guideline does not prescribe an anticoagulant treatment any more"

corresponding, in the high-level formalism we have used in Section 6, to the formula.

Example 1(b).

$\forall$run,$\Box$(done = anticoagulant_treatment_excluded $\rightarrow$ $\Box\neg$ (done = anticoagulant_treatment)).

The above temporal formula has to be verified on all runs. This is directly supported by SPIN, which however, accepts as input only propositional formulas (that is, formulas in which variable occurrences are not allowed) together with a binding of atomic propositions to conditions on variables. The property

Example 1(c).

$\Box$ (p $\rightarrow$ $\Box\neg$ q)

is then entered through the interface, together with the bindings "p = (done = anticoagulant_treatment_excluded)" and "q = (done = anticoagulant_treatment)".

SPIN automatically translates the input above into the corresponding automata, and verifies the property on all runs. In this example, the property was not true. As a consequence, SPIN reported as output a counter-example. The output directly generated by SPIN is quite complex (Fig. 6)

Example 1(d) presents a small part of the output generated by SPIN, which describes the Physician agent sending a message to the Guideline agent about the reliability of a patient's datum, and the Guideline agent working according to his answer (for more details about query action execution see Section 5.1).

In each line of Example 1(d) we find the number of the process corresponding to the agent, which is automatically generated by SPIN (e.g. *proc 3* in line 1 of Example 1(d)), its name (*Physician* in line 1 of Example 1(d)), the line of the Promela code being executed (*line 232* in line 1 of Example 1(d)), the file given as an input to SPIN (*pan_in* in line 1 of Example 1(d)), the number of the automaton state which the model-checker is visiting (*state 6* in line 1 of Example 1(d)), and finally the Promela code being executed (*LGfromPH_data!*`datum.D`,`datum.A`,`valid` in line 1 of Example 1(d)).

In particular, in Example 1(d), in first line the Physician agent sends a message to the Guideline agent, in the second line the Guideline agent receives the message itself, in the third line the Guideline agent tests the value of the *valid* variable; in the fourth line, since the execution of such a query action is finished, the variable *done* is set to numeric code of such an action, and in the fifth line, through the statement `goto`, the Guideline agent jumps to the next action.

Although the output generated by SPIN might not be easy to understand, by a simple preprocessing phase, we can:

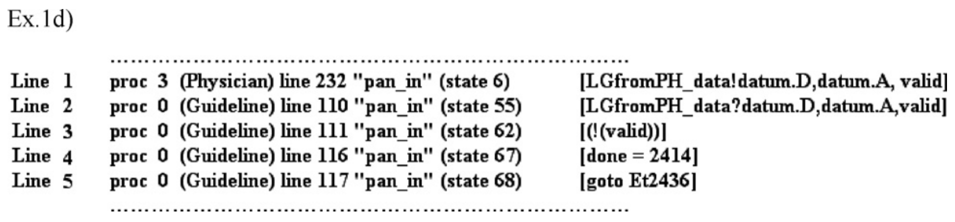
(i) select only the parts of the counter-example concerning assignments to the variable "done";
(ii) bind the numeric identifiers at the right-end of assignments to the name of the corresponding action in the GL (using a symbol table).

Thanks to this preprocessing phase, the output we report to end-users is not the one generated by SPIN as in Example 1(d), but is the path of actions in the GL which provides a counter-example to the property. Such actions are listed resorting to their names, as they were acquired in GLARE. Example 1(d) is thus converted as in Example 1(e) (where we present only the relevant part of the counter-example; observe that the complete counter-example is a path in the GL composed by almost 100 actions) (Fig. 7).

We think it is worth stressing that, although all the property check is performed by SPIN (thus using GL expressed in Promela, and properties expressed in LTL), the output is nevertheless easily interpretable by end-users, since it is directly related to the GLARE representation of the GL. As an easy extension, we plan to adopt GLARE's interface in order to graphically show to the end-user the path leading to the counter-example, thus making the output even more user-friendly. On the other hand, the problem of proposing a more user-friendly interface to express input properties requires substantial studies, as sketched in the section about future works.

As a consequence of the failure of the above check, the GL has been first modified by a team of domain experts, and such modifications have been applied also on the acquired GL, using GLARE acquisition interface.

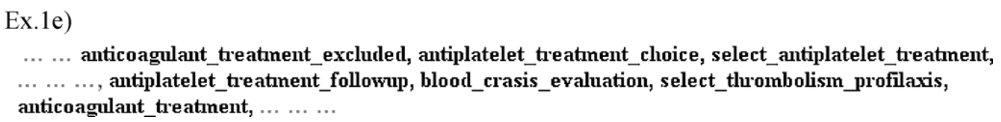
On the other hand, in the ischemic stroke GL example, the *contextualization phase* was skipped, since the GL was directly used in the hospital issuing it. However, contextualization would be important in case other hospitals would adopt such a guideline. For instance, a small hospital, in which the CT scanner is not available, could check whether there is at least a run in which CT scanner is not used (see Section 6.3).



```
Ex.1d)

          ………………………………………………………………………
Line  1   proc 3 (Physician) line 232 "pan_in" (state 6)    [LGfromPH_data!datum.D,datum.A, valid]
Line  2   proc 0 (Guideline) line 110 "pan_in" (state 55)   [LGfromPH_data?datum.D,datum.A,valid]
Line  3   proc 0 (Guideline) line 111 "pan_in" (state 62)   [(!(valid))]
Line  4   proc 0 (Guideline) line 116 "pan_in" (state 67)   [done = 2414]
Line  5   proc 0 (Guideline) line 117 "pan_in" (state 68)   [goto Et2436]
          ………………………………………………………………………
```

Figure 6. A part of the output generated by SPIN.



```
Ex.1e)
 … … anticoagulant_treatment_excluded, antiplatelet_treatment_choice, select_antiplatelet_treatment,
… … …, antiplatelet_treatment_followup, blood_crasis_evaluation, select_thrombolism_profilaxis,
anticoagulant_treatment, … … …
```

Figure 7. The relevant part of the GL path corresponding to the counter-example provided by SPIN.

Last but not the least, our verification approach has been used during the *execution* of the GL on specific patients (as an experiment, since currently the GLARE approach is not fully integrated within the hospital workflow). During the execution of a GL on a specific patient, a physician can use verification in order to check the applicability of the GL to the specific patient at hand. In particular, physicians can use verification in order to acquire new pieces of information to support their decisions (e.g., to check whether the GL contains one or more paths of actions which are eligible for the specific patient, and/or to discriminate between them).

For instance, during the execution of the ischemic stroke GL on a patient with atrial fibrillation in the ischemic stroke long-term prevention, the physician can choose between two different therapies: the anticoagulant therapy and the antiplatelet treatment. This choice is based on compliance and comorbidities of the patient. In general, the anticoagulant therapy is preferred, but the patient must be compliant to the treatment. For instance, in an elderly patient with some cognitive deficits the anticoagulant treatment is potentially at risk, so that antiplatelet drug treatment may be recommended. Additionally, in all cases, the allergies of the patient have to be taken into account. For instance, if the antiplatelet therapy is chosen, but the patient is allergic to acetilsalicilic acid (ASA) or previous gastroenteric ASA side effects have occurred, alternative drugs (for example ticlopidine, clopidogrel) should be used.

As a specific example, let us suppose to have an elderly patient with minor cognitive deficits, who is allergic to ASA. The physician, having to choose between the anticoagulant therapy and the antiplatelet treatment, may be inclined to select the latter therapy, and may use our verification approach as a support for her/his decision, to check whether there is at least a path in the GL stemming from the choice of the antiplatelet treatment in which no ASA administration is recommended.

In the high-level formalism we have used in Section 6, this property corresponds to the following formula:

$$\langle \exists \text{run},\ \Box(\text{done} = \text{antiplatelet\_choice} \rightarrow \Box(\text{done} \neq \text{ASA\_administration})) \wedge \Diamond(\text{done} = \text{antiplatelet\_choice})\rangle$$

SPIN does not directly support the above verification. To check if there is a run satisfying the LTL formula above, we give as input to SPIN the negation of the formula:

$$\neg(\Box(\text{done} = \text{antiplatelet\_choice} \rightarrow \Box(\text{done} \neq \text{ASA\_administration})) \wedge \Diamond(\text{done} = \text{antiplatelet\_choice}))$$

whose truth is verified on all the runs of the guideline. If the formula (1′) is not true on all runs of the guideline, SPIN provides a counter-example, namely, a run on which, when the antiplatelet treatment is chosen, no ASA administration is recommended.

8. Related work

The relevance of GLs in medical practice has been progressively increasing in time, leading to an evolution of the methodology for guideline definition. Historically, early guidelines were predominantly derived from unsystematically compiled opinions of experts based on clinical trials and mechanistic approaches (consensus-based medicine). Later on, evidence-based medicine (EBM) has become an essential component in the preparation of GL; the EBM attempts to provide a logical and convenient framework from which the quality and relevance of clinical studies (ideally randomized controlled trials) may be assessed in an unbiased manner. However, in many parts of the world, medicine is practiced in a context that is far (as concerns available resources, patterns of disease and cultural attitudes to health) from the environment in which GL in North America and Europe is developed. For example, a guideline developed in these countries on the basis of the latest and best evidence will be irrelevant in the developing world. Moreover, the clinician must translate all available evidence to the management of an individual patient and this stage requires that the evidence-based approach to be filtered through the opinion of doctors and journal editors (consensus-based medicine). In short, the development of useful GL today requires an integration of evidence with expert opinion and clinical expertise, as well as patients' values and preferences.

Our approach to computerized GL management stands also on the above reflections and, of course, GL verification takes into account, as very important elements, both the GL adherence to the best available evidence (deriving from meta-analysis of randomized, controlled clinical trials) and the possibility of GL contextualization.

Our work has started in the context of the Italian (two-years) project MIUR-PRIN 2003 "Logic-based development and verification of multi-agent systems" whose main objective was the development of logical and computational formalisms for the specification and verification of agents and their interactions. In this context, the activity on GL verification through model checking started as a case study [21]. Our work has then continued in the Italian two-year project MIUR-PRIN 2005 "Specification and verification of agent interaction protocols" which was specifically devoted to protocols and guidelines verification. In that project, two different proposals to guideline verification have been pursued: the one based on a computational logic framework [28], and the other one based on the use of model checking techniques [22,23]. Our choice of using LTL model checking, rather than CTL model checking, was mainly dictated by the fact that some of the logical formalisms used in the project were extensions of LTL, and that there is a consolidated on-the-fly LTL model-checker, SPIN [25]. In particular, SPIN input language Promela is a high-level language, well suited for the specification of asynchronous protocols, in which processes communicate by sending messages over channels. This was especially convenient/amenable for capturing our view of the guideline as an agent (a process) interacting with other agents (environment, physician, etc.). The work we present in this paper is an extension of the work in [22,23]. In both [22,23] we have proposed an architecture for integrating GLARE with the model-checker SPIN, and shown few examples of properties. In this paper, we have extended and systematized such an initial work along several lines: (i) we have generalized our methodology, which can be applied also to other guideline management systems; (ii) we have described our implementation, describing in detail both the representation of guidelines using Promela processes and the translator; (iii) we have proposed a systematic analysis of the different types of properties that can be verified; (iv) we have proposed detailed examples of how verification can be used, by whom, and when, and we have also analysed which are the possible effects of proving that a given desired property does not hold.

Considering the related approaches in the literature, to the best of our knowledge automatic verification of clinical guidelines has first been explored in [19], where a theorem proving approach is proposed to deal with the problem of protocol verification. This activity has been developed within the European projects Protocure (started in 2001) and Protocure II (started in 2006). Here, a medical protocol is modelled in the Asbru language as a hierarchical plan and then it is mapped to a specification in KIV, an interactive theorem prover for higher order logic. Properties are expressed in a variant of Interval Temporal Logic. Later on [2], has provided an evaluation of the feasibility of this approach based on the formalization and verification of the "jaundice" protocol and the "diabetes mellitus" protocol.

In the Protocure II project, besides the central theme of interactive verification of clinical guidelines with the theorem prover KIV, model checking techniques have been explored [20,29]. One of the motivations provided for this switch is that, in contrast to interactive verification, model checking is fully automatic. Moreover, in the Protocure project, they also noticed that, according to the formal semantics of Asbru, hierarchical plans that model the guideline are represented by state charts, namely, hierarchical state transition systems, which are highly suitable for model checking. The choice in Protocure II was for CTL model checking and for the tool SMV [30]. The Asbru model is translated into the input language of SMV model-checker by making use of a suitable abstraction which eliminates time. The compiler takes the algebraic specification of Asbru models in KIV as input and generates an SMV document. CTL model checking is used in the verification of a wide range of properties of guidelines modelled in Asbru, namely structural and medical properties [31]. In particular, in [20] properties of the jaundice protocol are formalized as ACTL formulas (that is, CTL formulas only allowing universal path quantifiers) [18].

The main difference between our approach and Protocure's one is that our approach is based on LTL temporal logic while Protocure's one is based on CTL temporal logic. The adoption of CTL (and ACTL) or LTL model checking allows for the verification of different temporal properties, as CTL and LTL are expressively incomparable (as well as ACTL and LTL): there are CTL formulas which cannot be expressed in LTL, as well as LTL formulas which cannot be expressed in CTL. For instance, the verification of the last property of Section 6.4 requires to check that the formula $\diamond$ (leukopenia_value = present $\wedge$ $\Box$ (done $\neq$ growth_factors_administration)) holds on all runs of the guideline. If there is a run on which this formula is not true, the counter-example provided by SPIN is a run of the guideline on which growth factors are administered whenever leukopenia is present, as required by the last property of Section 6.4. The above LTL formula cannot be expressed in CTL (or ACTL).

The debate between CTL and LTL has been recently revived by Vardi [32], who observed that LTL has advantages over CTL from the point of view of expressiveness, compositionality, property-specific abstraction and uniformity. According to Vardi, "CTL suffers from fundamental limitations as a specification language, all stemming from the fact that CTL is a branching time formalism" which makes the language sometimes unintuitive and hard to use. However, the debate about the relative merits of LTL and CTL is still open, and it is not at all the goal of our paper to contribute to such a debate. As a matter of fact, we believe that, in some sense, the two ways of expressing properties may be complementary, so that investigating the possibility of providing an integrated framework for GL verification in which both LTL and CTL are supported may be an interesting and fruitful issue for future research.

A further difference between our approach and Protocure one is due to the availability in SPIN of a higher level input language, as compared with the input language of SMV. The fact that Promela is well suited for modelling guidelines as processes interacting with their environment by exchanging messages over channels, substantially simplifies the task of providing a translation of GLARE guidelines into Promela code (which does not require intermediate levels of representation), as well as that of interpreting the results of verification. As a difference, in Protocure, an intermediate level of representation is used: the generation of an SMV model is obtained by translating the state charts modelling the behaviour of plans in Asbru formal semantics into a flat state transition system. Concerning the typology of the properties to be verified, as observed in [20] the model checking approach is well suited for the verification of structural and simple medical properties of the guideline, that normally do not require an incremental verification strategy. However, as shown in Section 6.6, some simple form of incremental verification can be performed, when the outcome of a verification step is used, interactively, to select the property to be checked in the next step.

Even if semantics is not the main focus of this paper, it is important to notice that, as a side effect of translating GLARE's guidelines into Promela, and adopting SPIN, we provide a formal semantics for GLARE's GLs, in the form of Büchi automata. Although there is a wide agreement about the importance of providing a clear semantic model for GLs, this issue has been faced in several quite different ways within the medical informatics community. In most cases, the semantics of guidelines has been only implicitly provided via an execution engine, which provides an interpretation of guidelines by executing them on specific patients. Considering explicit representations, a formal operational semantics has been provided for PROforma [33] via the definition of an abstract execution engine and of rules describing how the different guideline operations change the state of such an engine. On the other hand, in SAGE a mapping to standard terminologies and models (such as the virtual medical record) is advocated [34].

While the Asbru protocol representation language allows the semantics of guidelines to be defined through Asbru formal semantics [35], a logical semantics to guidelines has been provided in [28]. There, a graphical notation to express medical guidelines is introduced, which can be automatically translated to the logic-based formalism provided by the SOCS computational logic framework.

9. Conclusions and future work

GL verification is a demanding and difficult task, which can benefit from the adoption of advanced AI techniques. In this paper, we propose a general and flexible approach to such a task, based on the *integration* of a *computerized GL management system* with an *AI model-checker*. We first propose a general and system-independent *methodology*, proposing a modular architecture in which a *translation* module is used in order to loosely couple a computerized GL management system with a model-checker (so that both the system and the model-checker can be used as they are, without any modification). As a proof of concept, we then *instantiate* it by loosely coupling GLARE with the model-checker SPIN. Specifically, this step has involved an in-depth analysis about how GLs can be represented in the *specification* language Promela of the model-checker SPIN. Finally, we have also analysed *when* verification can be used within the GL life-cycle, and *what* types of properties can be verified using our approach. The ongoing experimentation on the guideline for ischemic stroke has shown that the automatic verification of properties in the model checking approach is able to discover inconsistencies in the guideline that had not been detected in advance by hand; in particular, we have identified an inconsistency related to the anticoagulant treatment administration, which is described in Section 7.

Concerning future work, we intend to extend the approach in this paper along four main directions.

First, in the current approach, we have partly neglected the treatment of temporal constraints between actions in the GLs. Temporal constraints are usually an intrinsic part of clinical guidelines, and, in GLARE, a lot of work has been done in order to represent them and to check their consistency [16,36]. However, here we have almost neglected temporal constraints in the translation from GLARE into Promela, also due to the fact that Promela does not provide any facility to cope with them. We plan to investigate to what extent GLARE's temporal constraints can be accounted for in a model checking approach.

Second, although knowledge engineers do usually participate to the acquisition phase, and may also cooperate in the contextualization phase, it is not realistic to assume that they can take part to the execution phase, when a user-physician applies a specific (computerized) guideline to a specific patient. Although LTL provides a very expressive formalism to represent the properties to be checked, a direct use of LTL (through the interface provided by the SPIN model-checker) may be quite difficult and challenging for user-physicians. In order to make our approach easy-to-use also in the execution phase as a future work, we envision the development of a user-friendly interface in order to facilitate the introduction of the properties to be verified. We believe that the analysis we have conducted in Section 6 of this paper is an important preliminary step to achieve such a goal. As a matter of fact, in Section 6 we have identified a set of useful and frequently used patterns of LTL queries. Given such a set, it is possible to define a user-friendly (possibly graphical) interface helping users to enter them. For instance, one might hide as much as possible to users the complexity of LTL formulas by supporting input incrementally, and through a set of queries useful to discriminate the quantifiers and modal operators to be used.[2] Of course, direct access to SPIN's interface to LTL must also be maintained, to allow users to express also complex properties not fitting the patterns. These topics will be object of major investigation in our future research activity. On the other hand, notice that the problem of making the output generated by the model-checker more readable to users is quite easy, since it just involves a light extension of GLARE's graphical interface in order to visualize the output of model checking, i.e., the pattern of actions for which a given property does not hold.

Third, we aim at developing a declarative and logical semantics for guidelines based on temporal logic specification.

Fourth, in [37] it has been shown that model checking techniques can also be fruitably exploited in order to provide guidelines with advanced critiquing facilities. We plan to further investigate such an issue in our future work.

Acknowledgments

The authors are grateful to the anonymous reviewers, whose very helpful comments helped them in restructuring and improving the paper significantly. The research was partially supported by the Italian project MIUR-PRIN 2005 "Specification and verification of agent interaction protocols".

References

[1] Overhage JM, Tierney WM, Zhou XH, McDonald CJ. A randomized trial of "corollary orders" to prevent errors of omission. Journal of the American Medical Informatics Association 1997;4(5):364–75.

[2] tenTeije A, Marcos M, Balser M, van Croonenborg J, Duelli C, van Harmelen F, et al. Improving medical protocols by formal methods. Artificial Intelligence in Medicine 2006;36(3):193–209.

[3] Musen MA, Rohn JA, Fragan LM, Shortliffe EH. Knowledge engineering for a clinical trial advice system: uncovering errors in protocol specifications. Bull Cancer 1987;74:291–6.

[4] Shahar Y, Miksch S, Johnson P. The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. Artificial Intelligence in Medicine 1998;14(1–2):29–51.

[5] Musen MA, Tu SW, Das AK, Shahar Y. Eon: a component-based approach to automation of protocol-directed therapy. Journal of the American Medical Informatics Association 1996;6(3):367–88.

[6] Shiffman RN, Karras BT, Agrawal A, Menco L, Chen R, Nath S. GEM: a proposal for a more comprehensive guideline document model using xml. Journal of the American Medical Informatics Association 2000;7(5):488–98.

[2] For instance, the interface may first ask whether the user wants to check whether a property holds for a specific case (run) or in all cases, to discriminate between the "for all runs" and "exists a run" cases. Then it may ask questions to identify the parameters and discriminate between the use of $\Box$ (e.g., "must the parameter hold for all states in the run?") and $\Diamond$ (e.g., "must the parameter hold for at least one state in the run?").

[7] Terenziani P, Molino G, Torchio M. A modular approach for representing and executing clinical guidelines. Artificial Intelligence in Medicine 2001;23(3): 249–76.

[8] Terenziani P, Montani S, Bottrighi A, Molino G, Torchio M. Applying artificial intelligence to clinical guidelines: the GLARE approach. In: Ten Teije A, Miksch S, Lucas P, editors. Computer-based medical guidelines and protocols: a primer and current trends. Amsterdam: IOS Press; 2008. p. 273–82.

[9] Peleg M, Boxwala AA, Ogunyemi O, Zeng Q, Tu S, Lacson R, et al. Glif3: the evolution of a guideline representation format. In: Proceedings of the American Medical Informatics Association annual symposium 2000; 2000. p. 645–9.

[10] Quaglini S, Stefanelli M, Lanzola G, Caporusso V, Panzarasa S. Flexible guideline-based patient careflow systems. Artificial Intelligence in Medicine 2001;22(1):65–80.

[11] Fox J, Johns N, Rahmanzadeh A, Thomson R. Disseminating medical knowledge: the PROforma approach. Artificial Intelligence in Medicine 1998;14(1–2):157–82.

[12] Gordon C, Christensen JP, editors. Health telematics for clinical guidelines and protocols. Amsterdam: IOS Press; 1995.

[13] Fridsma DB (Guest Ed.). Special issue on workflow management and clinical guidelines. Journal of the American Medical Informatics Association, 2001;1(22):1–80.

[14] Ten Teije A, Miksch S, Lucas P, editors. Computer-based medical guidelines and protocols: a primer and current trends. Amsterdam: IOS Press; 2008.

[15] Duftschmid G, Miksch S, Gall W. Verification of temporal scheduling constraints in clinical practice guidelines. Artificial Intelligence in Medicine 2002;25(2):93–121.

[16] Anselma L, Terenziani P, Montani S, Bottrighi A. Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. Artificial Intelligence in Medicine 2006;38(2):171–95.

[17] Peleg M, Tu S, Bury J, Ciccarese P, Fox J, Greenes RA, et al. Comparing computer-interpretable Guideline models: a case-study approach. Journal of the American Medical Informatics Association 2003;10(1):52–68.

[18] Clarke EM, Grunberg O, Peled DA. Model Checking. Cambridge, MA: MIT Press; 2000.

[19] Marcos M, Balser M, ten Teije A, VanHarmelen F, Duelli C. Experiences in the formalization and verification of medical protocols. In: Proceedings of 9th conference on artificial intelligence in medicine in Europe, lecture notes in computer science 2780. Berlin: Springer; 2003. p. 132–41.

[20] Baumler S, Balser M, Dunets A, Reif W, Schmitt J. A verification of medical guidelines by model checking—a case study. In: Proceedings of the 13th International SPIN Workshop, lecture notes in computer science 3925. Berlin: Springer; 2006. p. 219–33.

[21] Giordano L, Martelli A, Terenziani P, Bottrighi A, Montani S. A temporal approach to the specification and verification of Interaction Protocols. In: De Paoli F, Merelli E, Omicini A, editors. Proceedings of workshop from objects to agents 2005, Corradini. Bologna: Pitagora Editrice; 2005. p. 171–6.

[22] Terenziani P, Giordano L, Bottrighi A, Montani S, Donzella L. SPIN Model checking for the verification of clinical guidelines. In: ECAI 2006 workshop on AI techniques in healthcare: evidence-based guidelines and protocols. 2006. p. 101-6.

[23] Terenziani P, Giordano L, Bottrighi A, Montani S, Donzella L. Model checking for clinical guidelines: an agent-based approach. In: Proceedings of the American Medical Informatics Association annual symposium 2006; 2006. p. 289–93.

[24] Halpern JY, Vardi MY. Model checking vs. theorem proving: a manifesto. In: Allen JA, Fikes R, Sandewall E, editors. Proc. principles of knowledge representation and reasoning: proceedings of the second international conference. San Mateo, California: Morgan Kaufmann; 1991. p. 325–34.

[25] Holzmann GJ. The SPIN Model checker. Primer and reference manual. Boston, MA: Addison-Wesley; 2003.

[26] Montani S, Terenziani P. Exploiting decision theory concepts within clinical guideline systems: towards a general approach. International Journal of Intelligent Systems 1996;21(6):585–99.

[27] Allen Emerson E. Temporal and modal logic. In: van Leeuwen J, editor. Handbook of theoretical computer science. Formal models and semantics (B), vol. B. Cambridge, MA: Elsevier and MIT Press; 1990. p. 995–1072.

[28] Alberti M, Ciampolini A, Chesani F, Gavanelli M, Mello P, Montali M, Storari S, Torroni P. Protocol specification and verification by using computational logic. In: De Paoli F, Merelli E, Omicini A, editors. Proceedings of workshop from objects to agents 2005, Corradini. Bologna: Pitagora Editrice; 2005. p. 184–92.

[29] D4.3 Model checking of selected guideline properties. Technical Report of the project Protocure II. http://www.keg.uji.es/deliverables/D43-model-checking-guideline-properties.pdf [URL last accessed on 22/05/2009].

[30] McMillan KL. Symbolic model checking. Norwell, MA: Kluwer Academic Publisher; 1993.

[31] D4.2c Formal verification of selected guideline properties. Technical Report of the project Protocure II. http://www.keg.uji.es/deliverables/D42c-formal-verification-guideline-properties.pdf [URL last accessed on 22/05/2009].

[32] Vardi MY. Branching vs. linear time: final showdown. In: Proceedings of tools and algorithms for the construction and analysis of systems 2001, lecture notes in computer science 2031. Berlin: Springer; 2001. p. 1–22.

[33] Sutton DR, Fox J. The syntax and semantics of the PROforma guideline modeling language. Journal of the American Medical Informatics Association 2003;10(5):433–43.

[34] Parker CG, Rocha RA, Campbell JR, Tu SW, Huff SM. Detailed clinical models for sharable, executable guidelines. In: Fieschi M, et al., editors. Proceedings of

MedInfo (World Congress for Health Informatics) 2004. Amsterdam: IOS Press; 2004. p. 145–8.

[35] Balser M, Duelli C, Reif W. Formal semantics of Asbru—an overview. In: Proceedings of integrated design and process technology (IDPT-02). Society for Design and Process Science; 2002. p. 1–8.

[36] Terenziani P, German E, Shahar Y. The temporal aspects of clinical guidelines. In: Ten Teije A, Miksch S, Lucas P, editors. Computer-based medical guidelines and protocols: a primer and current trends. Amsterdam: IOS Press; 2008. p. 81–100.

[37] Groot P, Hommersom A, Lucas P, Serban R, ten Teije A, van Harmelen F. The role of model checking in critiquing based on clinical guidelines. In: Proceedings of 11th conference on artificial intelligence in medicine in Europe, lecture notes in computer science 4594. Berlin: Springer; 2007. p. 411–20.