ELSEVIER

Artificial Intelligence in Medicine 22 (2001) 23–41

Artificial Intelligence in Medicine

www.elsevier.com/locate/artmed

Knowledge-based verification of clinical guidelines by detection of anomalies

G. Duftschmid[a,*], S. Miksch[b]

[a]*Department of Medical Computer Sciences, University of Vienna, Spitalgasse 23, A-1090 Vienna, Austria*
[b]*Vienna University of Technology, Institute of Software Technology (IFS), Favoritenstrasse 9-11, A-1040 Vienna, Austria*

Received 31 May 2000; received in revised form 21 September 2000; accepted 8 November 2000

Abstract

As shown in numerous studies, a significant part of published clinical guidelines is tainted with different types of semantical errors that interfere with their practical application. The adaptation of generic guidelines, necessitated by circumstances such as resource limitations within the applying organization or unexpected events arising in the course of patient care, further promotes the introduction of defects. Still, most current approaches for the automation of clinical guidelines are lacking mechanisms, which check the overall correctness of their output. In the domain of software engineering in general and in the domain of knowledge-based systems (KBS) in particular, a common strategy to examine a system for potential defects consists in its verification. The focus of this work is to present an approach, which helps to ensure the semantical correctness of clinical guidelines in a three-step process. We use a particular guideline specification language called Asbru to demonstrate our verification mechanism. A scenario-based evaluation of our method is provided based on a guideline for the artificial ventilation of newborn infants. The described approach is kept sufficiently general in order to allow its application to several other guideline representation formats.

Keywords: Clinical guidelines and protocols; Verification; Medical plan management

1. Introduction

Within the last decade, public and private dissatisfaction about the perceived health and economical consequences of inappropriate medical care has significantly increased [5]. These perceptions stem from many sources including ceaselessly escalating health care costs, wide variations in medical practice patterns, and recognition that the obvious effect

* Corresponding author. Tel.: +43-1-40400-6696; fax: +43-1-40400-6697.
E-mail addresses: georg.duftschmid@akh-wien.ac.at (G. Duftschmid), silvia@ifs.tuwien.ac.at (S. Miksch).

0933-3657/01/$ – see front matter
PII: S0933-3657(00)00098-1

of some health services does not justify the efforts invested. Market pressures have further contributed to complicate the situation by driving medical organizations to increase productivity and to reduce costs, all without adversely affecting patient care.

One method that has been proposed as an answer to these problems is to adopt standard practice guidelines. The main goals that are pursued by the application of guidelines are the improvement of patient care's quality and the reduction of treatment costs. It could be shown that these desired effects can actually be reached, provided that the guidelines are properly formulated and followed [8].

In Section 2 we will give an overview of current approaches for the computerization of clinical guidelines and point out some techniques for their verification. After a short introduction of the Asbru language for the representation of guidelines in Section 3, we will present our verification method in Section 4. A scenario-based evaluation of our method will be provided in Section 5.

2. Related approaches to improve medical care

2.1. Automation of clinical guidelines

Being confronted with the laborious and inherently error-prone manual management of paper-based clinical guidelines, the interest in a computerized handling of the problem started to grow within the medical community. First attempts to automate guideline-based care were able to demonstrate a number of advantages over manual methods [13,30].

A common strategy among guideline applications is the so-called prescriptive approach, where an active interpretation of a preselected guideline is generated by the system, e.g. [6,11,20,25,29,31,32]. Another approach is the critiquing approach, where the system critiques the physician's plan rather than recommending a complete one of its own [2,33]. Finally, several approaches are based on the hypertext browsing of guidelines via the world wide web, e.g. [3,13].

2.2. Verification of clinical guidelines

One prerequisite for a broad acceptance and an efficient application of guidelines in the clinical domain is the guarantee of a high level of quality and reliability. However, as has been shown in numerous studies most clinical guidelines embody different kinds of semantical errors that compromise their practical value: guidelines are often vague and incomplete, they show serious omissions and unintentional ambiguities as well as unclear definitions, incompleteness, and inconsistency [15,16,18,26,27,29].

The situation is further complicated by the fact that the process of correcting such errors within a guideline is not a one-time matter. Rather this task is of repetitive nature as guidelines are subject to change. Changes may be effectuated by the evolution of a guideline to implement medical progress in the treatment of individual diseases. Another reason for the modification of a guideline, which will become effective even more frequently is addressed in [7–9]: generic, site-independent guidelines, designed to be

sharable between institutions, have to be adapted in most cases when applied by an individual organization according to the specific properties of the organization and of the patient to be treated. These adaptations may become necessary before the execution of a guideline due to resource or organizational limitations as well as particular patient characteristics. They may also become essential during the execution of a guideline due to unexpected events that arise in the course of patient care. In each of these cases, it has to be checked that the alterations done to the guideline do not affect its logical flow.

In contrast to the intensive efforts to develop new guidelines, the issue of providing mechanisms to ensure their overall correctness has been widely neglected thus far. However, as is already known from the domain of software engineering, an early identification of flaws within algorithmic knowledge is critically important [1]. In the domain of software engineering in general and in the domain of knowledge-based systems (KBS) in particular, a common strategy to examine a system for potential defects consists in its verification. We use the term *verification* as proposed in [12]: verification is defined as the sum of all processes, which attempt to determine whether a KBS does or does not satisfy its purely formal specifications. As will be shown in Section 4.2.2, we are concerned with specifications derived from formalizable concepts.

As already indicated within [5,14], approaches concerning the semantical verification of clinical guidelines are still rather rare. In the following we will give an overview of the few approaches concerning guideline verification, which have been published yet.

2.2.1. Decision-table techniques

One method for the verification and simplification of guidelines described in the literature consists in the logical analysis of guidelines by applying decision-table techniques [26,27]. Verification here is limited to two different properties, which are *completeness* and *consistency* of a guideline: a guideline is said to be complete when an action is defined for every possible value-combination of parameters used within the guideline. It is considered consistent, if each of its rules, consisting of a certain condition and an assigned action, is unique. If the latter property is violated, the authors distinguish between the three variants of *redundant*, *contradictory* or *conflicting* guidelines.

2.2.2. Examination of related tasks

Quaglini et al. [23] described how a guideline may be examined for logical correctness. As a formal representation model, they organize guidelines as sets of hierarchical tasks, which amongst others include activation conditions and subtasks. Subtasks may either be in AND-relation or in XOR-relation, which means that their parent task completes either after all subtasks have completed (AND), or after one and only one subtask has completed (XOR). To show a guideline's correctness Quaglini and co-workers check for *completeness* and *coherence*: Merging Shiffman and Greenes's concepts of completeness and conflict [27], Quaglini and co-workers consider a guideline complete, if the activation-conditions of each set of subtasks in XOR-relation are defined in a way that for every combination of variable values there is exactly one task activated by that combination. Checking a guideline for coherence means to look for conjunctive subtasks, which exclude each other because of incompatible activation conditions.

2.2.3. Condition checking based on semantic constraints

In [17] a system called Commander is described, which helps to verify a clinical guideline by checking its conditions similar to [27]. In contrast to Shiffman and Greenes, the authors ignore the *consistency* property in their verification process and exclusively concentrate on examining the *completeness* of a guideline. Hereby, they incorporate semantic constraints to reduce the combinations of variable values to consider.

2.2.4. Dynamic testing methods

In [28] an approach for the dynamic analysis of the ONCOCIN knowledge-base (KB) is described, the latter being used for the representation of cancer therapy guidelines. By using the ScriptGen system, the oncologist is given the possibility to generate certain sets of test cases, which allow for the testing of selected paths within a guideline. Hereby, instances are searched for in the KB whose behavior deviates from that dictated by the specification, which is the corresponding cancer therapy guideline. Typical defects are rules that fire erroneously, rules that contain errors in parameter domains and missing rules.

2.2.5. Verification approaches in the domain of KBS

Looking for general verification methods from the domain of KBS that may be useful, we found some parallels to our case: a substantial number of verification approaches in the field of KBS is based on the detection of flaws in rule-bases, e.g. [19,21,22,34]. As we will demonstrate in Section 4.2.1, a relationship can be established between a rule-base and a clinical guideline. This principally enables the reuse of work done for the verification of rule-based systems. However, existing work is far from being sufficient for a profound verification of clinical guidelines: techniques, which are reusable in our case, are too generic to allow the incorporation of those guideline-specific properties, which we consider essential for our verification processes.

3. The Asgaard/Asbru project

Most existing approaches for the automation of clinical guidelines suffer from at least one significant limitation: they require a complete modeling of the guideline up to its smallest grained components during design time and do not support site- or patient-specific adaptation in response to a changed environment or unexpected events. These approaches are lacking the types of knowledge, which would be essential for the purpose of guideline adjustment, or do not provide them in a structured format.

The time-oriented, intention-based plan-representation language *Asbru* [16], developed within the *Asgaard* project [24], offers such support. Clinical guidelines coded in the Asbru language are organized within a *plan-specification library.* In the following the term *plan* is used to designate a clinical guideline, coded in the Asbru language. By providing a formal description of a plan's *intention*, Asbru satisfies an essential demand for efficient guideline transformation [8]: the structured definition of a plan's intention allows it to be adapted while remaining faithful to its original aim.

Our verification approach, presented in Section 4, further supports Asbru's ability of plan modification: before any plan can be released that has been altered based on

compatible intentions, it must be verified amongst others that it is free of logical inconsistencies. The subject of determining compatibility of intentions represents another interesting research issue but is not addressed in this paper.

3.1. Components of Asbru

In Asbru a clinical guideline is modeled as a set of hierarchical plans. Each plan is identified by a unique name and consists of a set of arguments, including a *time annotation* that represents the temporal scope of the plan, and five elementary components. These components are *preferences*, *intentions*, *conditions*, *effects* and a *plan body*. Each plan may contain any number of subplans within its plan body, which may themselves be decomposed into sub-subplans. In the following, we will call such a tree structure of plans, starting from one root plan down to its leaf plans, a *plan hierarchy*. During execution time, the system interpreter attempts to decompose each plan into its subplans until an undecomposable leaf plan is found. Such atomic plans are called *primitive plans* and are transferred to a suitable agent for their execution.

Intentions, world states, actions/plans and effects are durative. Therefore, plan states and corresponding transition criteria are incorporated in Asbru to cope with the time-oriented environment. In this paper we will particularly focus on the verification of these transition criteria, which are expressed by means of the *condition* component.

3.2. Plan states and state-transition criteria

A set of eight different *plan states* is used to describe the actual state of a plan during plan selection and plan execution. Seven different conditions build the *state-transition criteria*, controlling transitions between neighboring plan states. Fig. 1 shows Asbru's *model of plan states* (MPS), a deterministic, finite-state automaton. It illustrates the sequence of possible states of a plan and the corresponding conditions shown above the arrows.

Asbru provides seven different conditions:

1. *filter-preconditions* need to hold initially if the plan is applicable, but can not be achieved;
2. *setup-preconditions* need to be achieved to enable a plan to start;
3. *activate-condition* determines if the plan should be started manually or automatically;
4. *suspend-conditions* determine when an `activated` plan has to be interrupted;
5. *abort-conditions* determine when an `activated` or `suspended` plan is terminated unsuccessfully;
6. *complete-conditions* determine when an `activated` plan is terminated successfully;
7. *reactivate-conditions* determine when a `suspended` plan can be continued.

The plan states, which are stopping the execution of a plan successfully or unsuccessfully (`rejected`, `completed`, `aborted` and `suspended` states), may be propagated in both directions of the plan hierarchy: the parent plan always propagates these plan states to its children, whereas a child plan propagates them to its parent plan if the child belongs to its parent's *continuation set* (see Section 4.2.1).

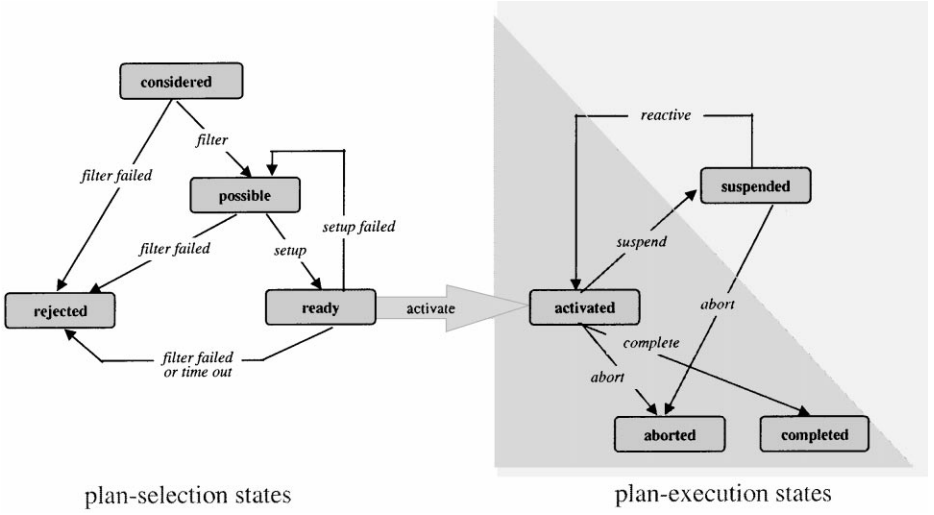

Fig. 1. Model of plan states — states and transition criteria during plan-selection (left) and plan-execution phases (right).

4. Verification of Asbru plans

We developed a partial verification method that aims at the identification of flaws within a guideline [4]. Hereby, we reuse existing verification work as the basis of domain-independent flaws, e.g. [21]. We further extend it by incorporating guideline-specific knowledge for the detection of flaws, which are characteristic for the domain of guideline-based care. The required knowledge is assumed to be available in a suitable KB component (see Section 4.2.3).

4.1. Methodology

Our verification approach examines the components of every Asbru plan and all its subplans for the existence of several anomalies, which indicate violations of corresponding specifications. The concept of anomalies is adopted from [22], where anomalies are defined as symptoms of probable errors. Our goal is to arrive at *meaningful* plans instead of totally correct plans. A plan is called meaningful, if it does not contain any anomalies, which would violate one of our specifications. We distinguish three levels of anomalies according to their locality (see Table 1).

The described approach is based on the hierarchical organization of clinical guidelines within the Asbru language. This type of structuring is also common in a wide range of other guideline representations, e.g. [6,11,23,31,32]. Therefore, our approach may also be applicable to numerous guideline models, other than the Asbru language.

The properties examined by our verification method are of static nature, which means that we will not have to execute any Asbru guideline in order to verify them.

Table 1
Three different levels of plan-verification: detect anomalies within single components (level 1), single plans (level 2) and whole plan hierarchies (level 3)

Decomposition	Detect anomalies within three levels		
	Level 1	Level 2	Level 3
Plan A			
Subplan A_a			
Component A_1	√		
...	√		
Component A_m	√	√	
...			√
Subplan A_x			
Component X_1	√		
...	√		
Component X_n	√	√	

4.1.1. Algorithm

The three levels are verified in a bottom-up fashion: first, level 1 is examined, which means that every single component of a plan is checked for anomalies within its own scope. The goal of checking level 2 is to detect anomalies, which result from dependencies between two or more components of a single plan. In level 3, the whole plan hierarchy is finally checked for anomalies that may originate from dependencies between two or more plans of the hierarchy. As guideline adaptations, in response to organizational or contextual issues, will primarily involve the exchange of whole plans, verification level 3 will be of particular importance in these cases. Fig. 2 shows an exemplary implementation of our method that is initiated by sending the message *verifyHierarchy* ( ) to all root-plans of the library.

We have shown in [4] that, whereas the complexity of checking levels 1 and 2 grows linearly with the number of plans, it is exponential for checking level 3 in the general case.

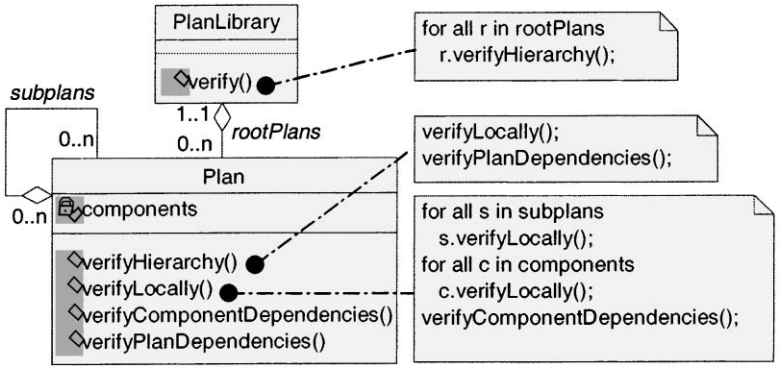

Fig. 2. Two classes with methods, relevant for three levels verification.

We have demonstrated, however, that level 3 can be verified with polynomial effort under the assumption that the size of examined component sets is limited by a suitable constant.

4.2. Verification of plan conditions

Any kind of guideline or plan, regardless of how it is modeled, needs a mechanism to control the sequence of its proposed actions. This mechanism is usually implemented through conditions. In the following, we will practically apply our verification method to illustrate the examination of Asbru conditions. Our considerations will be based on the existence of the following three features, incorporated in the Asbru language:

- conditions to control state transitions;
- a generic Model of Plan States;
- a hierarchical organization of plans.

4.2.1. Mapping plans to rule bases

Preece et al. gave a detailed summary of anomalies, that can occur in rule bases [21]. In the following we will show, how Asbru's Model of Plan States and every single plan can be transformed to a RB. We can then refer to [21] in formulating the specifications for plan conditions. Some of the anomalies we address, especially those which are more general in nature, are directly derived from [21]. These existing anomalies have been complemented with additional ones, which result from specifics of the Asbru language and do not have an equivalent [21].

After some definitions, we will first describe the complete and generic RB MPS for the Asbru's Model of Plan States (see Fig. 1). Its rules specify in which sequence the conditions of a plan are considered. The RB MPS is preset by the Asbru language, assumed to be initially verified and cannot be changed by the user. Therefore, it does not have to be verified further.

The second RB we present is called *PC*, and contains the plan conditions of all Asbru plans. It can be divided into rule bases PC_i for each plan, thus $PC = PC_1 \cup \cdots \cup PC_n$. All rules in *PC* have in common that they may only contain consequences, which are elements of *ConditionSet* (see below). As the rule bases PC_i are to be implemented by the user, they will be the target of our verification method.

4.2.1.1. Definitions. Assume *pl* and *pa* are parameters for entities "plan" and "patient".

PlanSet = set of all plans in the Asbru library.
PatientSet = set of all patients, to whom a plan may be applied.
ConditionSet = {filter(pl, pa), setup(pl, pa), activate(pl, pa), suspend(pl, pa), reactivate(pl, pa), abort(pl, pa), complete(pl, pa)} /* correlates with arrows in Fig. 1 */.
FinalStateSet = {rejected(pl, pa), aborted(pl, pa), completed(pl, pa)}.
Rule $R = L1 \wedge \cdots \wedge Ln \rightarrow M$; antec[1]$(R) = \{L1, \ldots, Ln\}$; conseq[2]$(R) = M$.
SS_i = set of all subplans of plan *i*.

[1] Abbreviation for *antecedent*.
[2] Abbreviation for *consequent*.

CS_i = *continuation set* of plan i. This is a subset of SS_i, containing all subplans, relevant for the completion of plan i: if a plan has subplans, some of them may need to complete as a prerequisite for the completion of plan i itself. The set of relevant subplans is defined in the parent plan.

A hypotheses H can be *inferred* from a *RB* for some environment E, if H is a logical consequence of supplying E as input to RB, formally: *infer* (H, RB, E) *iff* $(RB \cup E) \vdash H, E \in PatientSet$.

A hypotheses H is *inferable* from a *RB* if there is some environment E such that H can be inferred from RB for E, formally: *inferable*(H, RB) *iff* $(\exists\, E)$ *infer* (H, RB, E), $E \in PatientSet$.

A rule $R \in RB$ *fires* for some environment E, if the antecedent of R is a logical consequence of supplying E as input to *RB*, formally: *fire*(R, RB, E) *iff* $(\exists\, \sigma)$ $(RB \cup E) \vdash antec(R)\sigma$, $E \in PatientSet$.

A rule $R \in RB$ is *fireable* if there is some environment E such that R fires for E, formally: *fireable*(R, RB) *iff* $(\exists\, E)$ *fire*(R, RB, E), $E \in PatientSet$.

4.2.1.2. Rule bases MPS and PC. Table 2 shows the RB MPS. The order of the rules is important to determine which and when rules will be fired. The choice of rule ordering is oriented towards the state-transition criteria (compare Fig. 1).

Table 3 gives an example for a simplified version of a certain PC_k, a RB for all conditions of the plan *GDM-TYPE II* to treat noninsulin-dependent gestational diabetes mellitus in patients with normal blood-glucose levels.

In order to get a complete model for the execution control of a certain plan i, we have to unify its conditions base PC_i with the generic RB for Asbru's Models of Plan States, formally $MPS \cup PC_i$.

To handle the anomalies, which may occur within a RB $MPS \cup PC_i$, it is first necessary to outline the specific properties of this RB.

- We mentioned that we are going to map all conditions of Asbru plans to rules of the form $L1 \wedge \cdots \wedge Ln \rightarrow M$. As these rules may only contain conjunctive literals, we will have

Table 2
RB MPS representing the generic model of plan states

$R1$	Considered(pl, pa) /* starting state for each plan */
$R2$	Considered(pl, pa) $\wedge$ filter(pl, pa) $\rightarrow$ possible(pl, pa)
$R3$	Considered(pl, pa) $\wedge \neg$ filter(pl, pa) $\rightarrow$ rejected(pl, pa)
$R4$	Possible(pl, pa) $\wedge$ setup(pl, pa) $\rightarrow$ ready(pl, pa)
$R5$	Possible(pl, pa) $\wedge \neg$ filter(pl, pa) $\rightarrow$ rejected(pl, pa)
$R6$	Ready(pl, pa) $\wedge \neg$ filter(pl, pa) $\rightarrow$ rejected(pl, pa)
$R7$	Ready(pl, pa) $\wedge \neg$ setup(pl, pa) $\rightarrow$ possible(pl, pa)
$R8$	Ready(pl, pa) $\wedge$ activate(pl, pa) $\rightarrow$ activated(pl, pa)
$R9$	Activated(pl, pa) $\wedge$ abort(pl, pa) $\rightarrow$ aborted(pl, pa)
$R10$	Activated(pl, pa) $\wedge$ complete(pl, pa) $\rightarrow$ completed(pl, pa)
$R11$	Activated(pl, pa) $\wedge$ suspend(pl, pa) $\rightarrow$ suspended(pl, pa)
$R12$	Suspended(pl, pa) $\wedge$ reactivate(pl, pa) $\rightarrow$ activated(pl, pa)
$R13$	Suspended(pl, pa) $\wedge$ abort(pl, pa) $\rightarrow$ aborted(pl, pa)

Table 3
RB PC_k representing the conditions of plan *GDM-TYPE II*

Female(pa) ∧ Pregnant(pa) → filter(GDM-TYPE II, pa)
Test_available(glucose-tolerance-test, pa) → setup(GDM-TYPE II, pa)
Activate(GDM-TYPE II, pa) /* automatic activation */
Delivered(pa) → complete(GDM-TYPE II, pa)
State(blood-glucose, high, pa) → suspend(GDM-TYPE II, pa)
State(blood-glucose, normal, pa) → reactivate(GDM-TYPE II, pa)
Insulin-indicator(pa) → abort(GDM-TYPE II, pa)

to find correct substitutions for the disjunctions, allowed within Asbru conditions. This will be done by splitting a rule at each disjunction, thereby creating a new rule with identical consequent for each disjunction. As an example, rule $(L1 \wedge L2) \vee (L3 \wedge L4) \rightarrow M$ will be split into two rules $L1 \wedge L2 \rightarrow M$ and $L3 \wedge L4 \rightarrow M$.

- The set of possible consequences of rules we have to consider is small: as we mentioned before, only the rule bases *PC* is modifiable by the user and only the conditions $C \in ConditionSet$ should be addressed by rules within it. This means that *PC* should only contain rules R with $conseq(R) = C$. As Asbru's Model of Plan States defines seven different conditions, each PC_i will usually contain around seven rules with different consequences, even though the actual number of rules may be slightly lower or higher: it may be lower, as the conditions are optional plan elements. It may be slightly higher, as a condition may include disjunctions and would then be split into several rules. Evidently, it will be easy to guarantee that *PC* only contains rules referring to valid Asbru conditions: one might provide some sort of input support to the user or even apply manual verification due to the small number of rules.
- The amount of rules we have to consider when checking anomalies within one plan is limited to the scope of the plan's hierarchy: plans of different hierarchies are independent, no anomalies can result from relationships between their rules.
- The possible sequences (rule ordering), in which the rules of any PC_i are considered, is preset by the rule bases MPS. In contrary to ''general'' RB, we can, therefore include the order of inference into our considerations. This allows us to include some dynamic aspects of Asbru plans into our static verification process: although, we will not actually execute a plan to verify it, RB MPS gives us the opportunity to take the control flow of a plan during execution into account.

4.2.2. Anomalies

In the following, we will list all anomalies concerning Asbru conditions and the corresponding specifications they violate. The anomalies are organized according to the levels in which they may occur (compare Table 1). The underlying specifications may be seen as an extension of existing verification work, achieved through the integration of Asbru specific language features such as its Model of Plan States (see Fig. 1).

Apart from their potential occurrence in original, generic guidelines, anomalies may be caused by inadequate specialization of guidelines: anomalies may be introduced by adaptation of guidelines to the specific characteristics of a receiving patient, e.g. when tuning certain thresholds of single conditions for a particularly sensitive patient (level 1).

This may also damage the correct interaction with other conditions within the same plan (level 2). Level 3 anomalies may be caused by guideline modifications due to organizational limitations: here a typical strategy will be to exchange whole, unsuitable plans with others that pursue compatible goals. Such replacements may interfere with the correct interaction with other plans of the same hierarchy.

4.2.2.1. Level 1: unsatisfiable conditions. Any single condition being part of a plan must have a chance to be satisfied during execution of the plan in order to have an influence on the plan's behavior. Referring to the above rule bases, we equivalently specify that each rule of *PC* must be fireable.

Formally: $fireable(R, PC) \forall R \in PC$.

The violation of this specification is a special case of Preece and co-workers *redundant rule* [21]. It may for example originate from medically implausible conditions (e.g. domain or type violations), corresponding to the *illegal attribute values anomaly* described in [19]. Another possibility would be a condition that contains a conjunction of incompatible parameter values.

Example: $male(pa) \wedge pregnant(pa) \rightarrow filter(PlanX, pa)$.

4.2.2.2. Level 1: redundant parameter-value pairs within conditions. Asbru conditions may contain conjunctions and disjunctions of parameter-value pairs. Each of them should check for some additional data, redundant tests would not make sense. This means we should avoid conditions containing several identical or entailing parameter-value pairs.

A conjunction of entailing parameter-value pairs within a condition is equivalent to the *redundant literal* anomaly defined in [21]. We stated that each rule is of the form $R = L1 \wedge \cdots \wedge Ln \rightarrow M$.

Then we request formally: $\neg ((Li \rightarrow Lj) \wedge (i \neq j) \forall (R \in PC;\ 1 \leq i, j \leq n))$.

We mentioned before that a condition consisting of disjunctive parameter-value pairs would be split into several rules in RB *PC*, one split for each disjunction. We will, therefore have to check within each PC_i that there is no pair of rules subsuming each other, corresponding to the *subsumed rule* anomaly defined within [22].

Formally: $\neg ((\exists \sigma)\ (R \rightarrow R'\sigma) \forall (R, R' \in PC_i;\ 1 \leq i \leq n))$.

Below, a possible representative of this anomaly is shown.

Example: $higher(cholesterol, 200, pa) \wedge higher(cholesterol, 220, pa) \rightarrow filter(PlanX, pa)$.

4.2.2.3. Level 2: unreachable, valid sequence of plan states. Any sequence of plan states, which defines a valid path according to Asbru's Model of Plan States, must also statically be possible within a single plan. Consequently, all conditions belonging to a valid path of plan states must be satisfiable. Otherwise, one or more states of the plan would not be reachable.

For every rule contained in every PC_i, we demand that there must exist a patient for whom the rule is fireable, considering all rules in PC_i, which have fired before for the same patient. For each rule R in PC_i, the set of rules, which must fire before considering R is defined through *MPS*. We can, therefore specify that for each plan i there must exist an environment E, such that each consequence in *MPS* can be inferred by supplying E as input to $MPS \cup PC_i$.

Formally: $((\exists\ \sigma)\ \mathit{inferable}(\mathit{conseq}(R)\sigma, \mathit{MPS} \cup \mathit{PC}_i)\forall(R \in \mathit{MPS};\ 1 \leq i \leq n))$.

If any two conditions of a valid state path happen to be incompatible, the plan state of the later checked condition cannot be reached, and the plan contains an anomaly.

Example: plan state `ready` is not inferable for PlanX below.

R_x: $\mathit{male}(pa) \rightarrow \mathit{filter}(\mathit{PlanX}, pa)$;

R_y:$\mathit{pregnant}(pa) \rightarrow \mathit{setup}(\mathit{PlanX}, pa)$.

4.2.2.4. Level 2: ambiguous state transition. Each time a state transition takes place during the execution of a plan, it must be clear which will be the target state of the transition, as a plan may only have one active state at a time. Conditions, that fork the state path within the Model of Plan States (e.g. the `complete−`, `suspend−` and `abort− condition` in Fig. 1) are the source of potential problems: if more than one of these conditions becomes true at the same time, the successor plan state will be ambiguous.

An intuitive solution would be to request *mutual exclusion* from forking-conditions. However, this strategy is probably unrealistic: we would impose a too harsh restriction on a plan designer by demanding that each set of forking-conditions had to be mutually exclusive. Especially, if the conditions are complex, it will be annoying for the designer to be forced to design the conditions in a way that they exclude each other. A more relaxed strategy would be to demand that forking-conditions should be independent in a way that their concurrent satisfaction is at least not statically foreseeable. In other words we could specify that the antecedents of forking-conditions must not entail each other. As this demand does not rule out a concurrent satisfaction of two forking-conditions, we must provide a heuristic to determine the proper target-state in a conflict situation of this kind.

For the purpose of the definition of the following anomaly, we will interpret all successor states of a state path fork as *semantic constraint expressions*, meaning that their concurrent occurrence does not make semantic sense (see Section 4.2.3).

$\mathit{ConstraintsSet} = \{(\mathit{possible}(pl, pa), \mathit{rejected}(pl, pa)), (\mathit{ready}(pl, pa), \mathit{rejected}(pl, pa)), (\mathit{suspended}(pl, pa), \mathit{aborted}(pl, pa)), (\mathit{suspended}(pl, pa), \mathit{completed}(pl, pa)), (\mathit{completed}(pl, pa), \mathit{aborted}(pl, pa)), (\mathit{activated}(pl, pa), \mathit{aborted}(pl, pa))\}$.

In order to avoid statically predictable, ambiguous state transitions, we can request that there must not be any pair of rules R and R' in $\mathit{MPS} \cup \mathit{PC}_i$, such that the antecedent of R entails the antecedent of R', and their consequents infer a semantic constraint expression from *ConstraintsSet*.

Formally: $\neg\ ((\exists\ \sigma)\ \ (\mathit{antec}(R)\sigma \rightarrow \mathit{antec}(R')\sigma) \wedge (\{\mathit{conseq}(R)\sigma, \mathit{conseq}(R')\sigma\} \in \mathit{ConstraintsSet})\forall(R, R' \in \mathit{MPS} \cup \mathit{PC}_i;\ 1 \leq i \leq n))$.

This corresponds to the *ambivalent rule pair* anomaly in [22]. Below, a possible representative of this anomaly is shown, where it is not clear whether PlanX should be suspended or aborted if the patient's bilirubin level is greater or equal 15.

Example: R_x: $\mathit{higher}(\mathit{bilirubin}, 5, pa) \rightarrow \mathit{suspend}(\mathit{PlanX}, pa)$;

R_y: $\mathit{higher\_equal}(\mathit{bilirubin}, 15, pa) \rightarrow \mathit{abort}(\mathit{PlanX}, pa)$;

R_z: $\mathit{lower\_equal}(\mathit{bilirubin}, 1, pa) \rightarrow \mathit{complete}(\mathit{PlanX}, pa)$.

4.2.2.5. Level 3: inability to complete. For the determination of a plan's ability to complete, it is necessary to check whether the `complete- condition` and all predecessor conditions can be satisfied for all plans, belonging to the plan's *continuation set*.

Using the abbreviation CS_i for the *continuation set* of plan i, we specify: for each plan i there must exist an environment E such that the consequence of rule $R10 \in MPS$ can be inferred by supplying the *same* E as input to $MPS \cup PC_k$ for all $k \in CS_i$.

Formally: $((\exists\, E \in PatientSet, \sigma)\quad (infer(conseq(R10)\sigma, MPS \cup PC_k, E))\forall(1 \leq i \leq n; k \in CS_i; R10 \in MPS))$.

We distinguish two scenarios that prevent a plan's completion:

- *Incompatible conditions within the plan itself:* this kind of anomaly has to be checked for in level 2 and is already covered by the unreachable, valid sequence of plan states anomaly.
- *Incompatible conditions within two or more plans that belong to the continuation set of the same plan:* all plans, belonging to the continuation set of a certain PlanX, have to complete as a prerequisite for the completion of PlanX. Therefore, no incompatibility must occur in a set of conditions, required to reach the `complete state` for all of these plans.

  Example: $CS_{PlanA} = \{PlanAa, PlanAb\}$ /* *continuation set of PlanA* */.

  R_x: $blood\text{-}group(pa, A) \rightarrow filter(PlanAa, pa)$;

  R_y: $blood\text{-}group(pa, B) \rightarrow filter(PlanAb, pa)$.

  The `filter − preconditions` of plans PlanAa and PlanAb, corresponding to rules R_x and R_y, are incompatible and both plans belong to the *continuation set* of their parent PlanA. Therefore, PlanA will not be able to complete.

4.2.2.6. Level 3: termination enforced by parent. As explained in Section 3.2, the plan states `rejected`, `aborted`, `suspended` and `completed` may be propagated from a plan to its subplans, thereby stopping the latter. This kind of overruling a plan's actual state should not be the ordinary case, and it should, therefore be assured, that the relevant conditions avoid such a situation. As state propagation might be deliberately applied in some cases, the violation of this specification is not considered an error, but a warning. In terms of our KB we specify:

SS_i = set of all subplans of plan i;
$RS = \{R3, R5, R6, R9, R10, R13\} \subseteq MPS$: rules with a consequent $\in FinalStateSet$;
$R11 \in$ MPS: rule with consequent $suspended(pl, pa)$.

Then we demand for each plan i: whenever the consequent of a rule R from FSR can be inferred for a certain environment E, then it must also be possible to infer the consequent of a rule R' from FSR for each of plan i's subplans for the same environment E. The same holds for rule $R11$ from MPS.

Formally: $infer(conseq(R)\sigma, MPS \cup PC_i, E) \rightarrow infer(conseq(R')\sigma, MPS \cup PC_k, E)\forall (1 \leq i \leq n; R, R' \in FSR; k \in SS_i; E, \sigma)$.

$infer(conseq(R11)\sigma, MPS \cup PC_i, E) \rightarrow infer(conseq(R11)\sigma, MPS \cup PC_k, E)\forall(1 \leq i \leq n; E)\forall(1 \leq i \leq n; R11 \in FSR; k \in SS_i; E, \sigma)$.

In the anomaly below, the `abort − conditions` of parent PlanA (rule R_x) does not entail the `abort − conditions` of child PlanAa (rule R_y). Consequently, there is a chance that the parent will override its child's true state with the state `aborted`.

Example: R_x: $higher(bilirubin, 5, pa) \rightarrow abort(PlanA, pa)$;

R_y: $higher(GOT, 22, pa) \rightarrow abort(PlanAa, pa)$.

4.2.3. Functionality of the verification knowledge base

We assume that our verification method has access to a domain-specific KB, which can be queried for the following information:

- *Incompatibility of conditions*. The KB defines which findings cannot occur simultaneously for the same patient. The concept of incompatibility is equivalent to the *semantic constraint expression* used in [21]: a semantic constraint expression is an expression $\{L1, \ldots, Ln\}$, which is interpreted as meaning that the simultaneous truth of $L1 \wedge \cdots \wedge Ln$ would not make semantic sense. For example, the set {blood-group-A(x), blood-group-B(x)} says that, for all x, x cannot have blood-groups A and B at the same time.
- *Entailment of conditions*. Entailment is not only restricted to conditions concerning the same parameters. In the medical domain there is a high number of dependencies between different parameters, that may be the source of non-trivial entailment (e.g. parameter *gender* and pregnancy-related parameters).
- *Attributes of medical parameters*. For each medical parameter that might be used in an Asbru plan, the KB contains information about its value domain and its type.

5. Scenario-based evaluation

After the theoretical foundation of our verification method has been illustrated, we will in the following describe a scenario of its application. Object of the analysis will be an exemplary guideline for the artificial ventilation of newborn infants, shown in Fig. 3.

The top-level plan is called infants respiratory distress syndrome therapy (*I-RDS-therapy*). It consists of four subplans that are decomposed into further subplans. Fig. 4 shows an excerpt of these plans, coded in Asbru and including different anomalies.

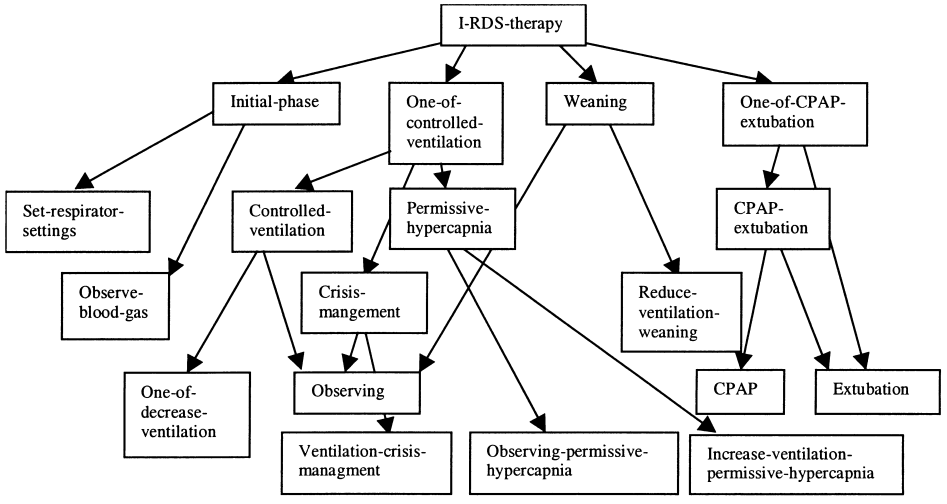

Fig. 3. Plan hierarchy of guideline for artificial ventilation of newborn infants, suffering from infant respiratory distress syndrome (I-RDS).

```
(PLAN one-of-controlled-ventilation
  (COMPLETE-CONDITIONS (PIP (< 20) I-RDS *now*))
  (DO-SOME-ANY-ORDER
    (controlled-ventilation)
    (permissive-hypercapnia)
    (crisis-management)
    CONTINUATION-CONDITION controlled-ventilation))
(PLAN controlled-ventilation
  (INTENTION:INTERMEDIATE-STATE
   (MAINTAIN STATE(BG) NORMAL controlled-ventilation *))
  (SETUP-PRECONDITIONS (PIP (<= 30) I-RDS *now*)
  (ABORT-CONDITIONS ACTIVATED
   (OR (PIP (> 30) controlled-ventilation *)
       (PIP (> 40) controlled-ventilation *)))
  (COMPLETE-CONDITIONS
   (FiO2 (<= 50) controlled-ventilation
     [[_, _], [_, _], [180 MIN, _], *self*])
   (f (<= 60) controlled-ventilation
     [[_, 2 WEEKS], [5 WEEKS, _], [_ , _], BIRTH]))
  (DO-ALL-SEQUENTIALLY
   (one-of-increase-decrease-ventilation)
   (observing)))
(PLAN permissive-hypercapnia
  (INTENTION:INTERMEDIATE-STATE
   (ACHIEVE STATE(BG) NORMAL controlled-ventilation
     [[_, _], [_, _], [_ , 30 Min], *self*] 0.9))
  (SETUP-PRECONDITIONS
   (PIP (>= 30) controlled-ventilation *now*)
  (SUSPEND-CONDITIONS (PIP (< 5) I-RDS *now*))
  (COMPLETE-CONDITIONS (PIP (< 25) I-RDS *now*))
  (DO-ALL-SEQUENTIALLY
   (increase-ventilation-permissive-hypercapnia)
   (observing-permissive-hypercapnia)))
```

Fig. 4. Fragment of I-RDS plans.

Anomalies, detected in checking-level 1:

Redundant parameter-value pairs in ABORT-CONDITIONS **of**

plan controlled-ventilation **:**

```
(PIP (> 40) controlled-ventilation *)
```

entails

```
(PIP (> 30) controlled-ventilation *)
```

Anomalies, detected in checking-level 2:

Target state of transition ambiguous in

plan permissive-hypercapnia **:**

```
(SUSPEND-CONDITIONS (PIP (< 5) I-RDS *now*))
```

entails

```
(COMPLETE-CONDITIONS (PIP (< 20) I-RDS *now*))
```

Anomalies, detected in checking-level 3:

Plan one-of-controlled-ventilation **may stop**

child plan controlled-ventilation **:**

```
(COMPLETE-CONDITIONS (PIP (< 20) I-RDS *now*)
```

does not entail

```
(COMPLETE-CONDITIONS
   (FiO2 (<= 50) controlled-ventilation
     [[_, _], [_, _], [180 MIN, _], *self*])
   (f (<= 60) controlled-ventilation
     [[_, 2 WEEKS], [5 WEEKS, _], [_,_],BIRTH]))
```

Fig. 5. Exemplary output of verifying plan-hierarchy *I-RDS-therapy*.

Fig. 5 shows an exemplary output for the analysis of the above plan-hierarchy that may be generated by the verifier.

6. Conclusion

Within this paper a method has been presented, which allows for the verification of clinical guidelines represented in a computer-readable format, in a three levels process. The ability of automated verification strongly supports the adaptation of generic guidelines to organization- or patient-specific characteristics, as the latter process is susceptible to the

introduction of flaws in the original guideline's logic. The verification of a clinical guideline is done by checking it for the occurrence of a certain set of predefined anomalies. In the identification process of these anomalies we oriented on one particular language for the representation of clinical guidelines, called Asbru. In this work we have focused on anomalies concerning the Asbru language element *condition*.

Our approach is reusable in two ways: first, it is applicable for the verification of numerous guideline-representation formats, other than Asbru. This is due to the fact that our method is based on a hierarchical organization of guidelines and the usage of conditions to control the guideline's execution flow. Both concepts are common in most current approaches. Second, our approach can be ported to other domains than guideline-based care, as the Asbru language is suitable for several areas of planning [16]. For the reuse of our verification approach in those other areas, it would only be necessary to provide the corresponding, domain-specific knowledge to adapt our KB component.

Another advantage of our method is given by the fact that it allows a significant limitation of the computational effort required. Instead of examining the whole guideline library for the detection of an anomaly, the search can be limited to a single guideline hierarchy. Besides the computational effort also the verification's complexity is reduced. This is reached by means of the information hiding process of decomposing a guideline into its components and performing stepwise, local verification. A final benefit of our organization of the verification process into three separate levels is revealed by the fact that it prepares us for an incremental verification process: adding a new, locally verified plan (levels 1 and 2) to an already verified plan hierarchy for example only requires the repetition of level 3 checks within the extended hierarchy.

After outlining the advantages of our approach we will also comment on its weaknesses: our verification method is designed for the detection of static anomalies within the guideline code. Although, we include dynamic aspects of Asbru plans into the verification process by considering the Asbru MPS, our method does not support the analysis of executing guidelines. A second limitation of our method, which is, however typical for all verification approaches that rely on anomaly detection, lies in its inability to guarantee a totally correct guideline. This shortage originates from the fact that the set of anomalies to be considered are identified in a heuristical process. Therefore, one can never be sure that really all anomalies that may possibly occur within a guideline have actually been handled.

However, even though we cannot provide a complete list of all different anomalies, it is undoubtedly useful to know that a certain range of anomalies will be uncovered by the described verification approach. By applying this strategy, we find ourselves in conformity with the domain of error-based testing, where the goal of running tests is not to show that the program is free from all errors but rather that the program is free from certain well-defined types of errors [10].

Acknowledgements

The authors thank Klaus Hammermüller, Robert Kosara, Andreas Seyfang and Yuval Shahar, who contribute to the development of the project. We are also grateful to Michael Balser, Wolfgang Reif, Annette ten Teije and Frank van Harmelen for

their valuable suggestions. The Asgaard project is supported by ''Fonds zur Förderung der wissenschaftlichen Forschung'' (Austrian Science Fund), P12797-INF.

References

[1] Adrion W, Branstad M, Cherniavsky J. Validation, verification and testing of computer software. Comput Rev 1982;14(2):159–92.
[2] Advani A, Lo K, Sahar Y. Intention-based critiquing of guideline-oriented medical care. In: Proceedings of the AMIA Annual Symposium 98 Orlando, FL, 1998, p. 483–7.
[3] Barnes M, Barnett G. An architecture for a distributed guideline server. In: Proceedings of the 19th Annual Symposium on Computer Applications in Medical Care. Philadelphia: Hanley and Belfus, 1995. p. 233–7.
[4] Duftschmid G. Knowledge-Based Verification of Clinical Guidelines by Detection of Anomalies, PhD Thesis. Vienna University of Technology, Vienna, 1999.
[5] Field M, Lohr K. Clinical Practice Guidelines: Directions for a New Program. Institute of Medicine, Washington (DC): National Academy Press, 1990.
[6] Fox J, Johns N, Rahmanzadeh A. Protocols for medical procedures and therapies: a provisional description of the PROForma language and tools. In: Proceedings of the Sixth Conference on Artificial Intelligence in Medicine Europe (AIME), Grenoble, France, 1997, p. 21–38.
[7] Fridsma D. Representing the work of medical protocols for organizational simulation. In: Proceedings of the AMIA Annual Symposium. Orlando, 1998. p. 305–8.
[8] Fridsma D, Gennari J, Musen M. Making generic guidelines site-specific. In: Proceedings of the 20th Annual Symposium on Computer Applications in Medical Care. Philadelphia: Hanley and Belfus, 1996. p. 597–601.
[9] Fridsma D, Thomsen J. Representing medical protocols for organizational simulation: an information processing approach. Comput Math Org Theory 1998;4(1):71–95.
[10] Hamlet R. Special section on software testing. Commun ACM 1988;31:662–7.
[11] Herbert S, Gordon C, Jackson-Smale A, Renaud S. Protocols for clinical care. Comput Methods Progr Biomed 1995;48:21–6.
[12] Laurent JP. Proposals for a valid terminology in KBS validation. In: Proceedings of the 10th European Conference on Artificial Intelligence (ECAI-92), Vienna, Austria, 1992, p. 829–34.
[13] Liem E, Obeid J, Shareck P, Sato L, Greenes R. Representation of clinical practice guidelines through an interactive world-wide-web interface. In: Proceedings of the Annual Symposium on Computer Applications in Medical Care (SCAMC-95). New Orleans (LA), 1995.
[14] McCormick K, Moore S, Siegel R. Clinical practice guideline development: methodology perspectives. Rockville (MD): AHCPR Publication No. 95-0009, Agency for Health Care Policy and Research, 1994.
[15] McDonald C, Overhage J. Guidelines you can follow and trust: an ideal and an example. J Am Med Assoc 1994;271(11):872–3.
[16] Miksch S, Shahar Y, Johnson P. Asbru: A task-specific, intention-based and time-oriented language for representing skeletal plans. In: Proceedings of the Seventh Workshop on Knowledge Engineering: Methods and Languages (KEML-97). UK: Milton Keynes, 1997.
[17] Miller DWJ, Frawley SJ, Miller PL. Using semantic constraints to help verify the completeness of a computer-based clinical guideline for childhood immunization. Comput Methods Progr Biomed 1999;58(3):267–80.
[18] Musen M, Rohn J, Fagan L, Shortliffe E. Knowledge engineering for a clinical trial advice system: uncovering errors in protocol specification. Bull du Cancer 1987;74(291):296.
[19] Nguyen T, Perkins W, Laffey T, Pecora D. Knowledge base verification. Artif Intell Mag 1987;2(2):69–75.
[20] Ohno-Machado L, Gennari J, Murphy S, Jain N, Tu S, Oliver D, Pattison-Gordon E, Greenes R, Shortliffe E, Barnett G. The guideline interchange format: a model for representing guidlines. J Am Med Assoc 1998;5(4):357–72.
[21] Preece A, Batarekh A, Shinghal R. Verifying rule-based systems. Knowledge Eng Rev 1992;7(2):115–41.
[22] Preece A, Shinghal R. Foundation and application of knowledge base verification. Int J Intell Syst 1994;9(8):683–702.

[23] Quaglini S, Saracco R, Stefanelli M, Fassino C. Supporting tools for guideline development and dissemination. In: Proceedings of the Sixth Conference on Artificial Intelligence in Medicine Europe (AIME), Grenoble, France, 1997, p. 39–50.
[24] Shahar Y, Miksch S, Johnson P. The Asgaard project: a task-specific framework for the application and critiquing of time-oriented clinical guidelines. Artif Intell Med 1998;14:29–51.
[25] Sherman E, Hripcsak G, Starren J, Jender R, Clayton P. Using intermediate states to improve the ability of the Arden syntax to implement care plans and reuse knowledge. In: Proceedings of the Annual Symposium on Computer Applications in Medical Care (SCAMC-95), New Orleans, LA, 1995, p. 238–42.
[26] Shiffman R. Representation of clinical practice guidelines in conventional and augmented decision tables. J Am Med Inform Assoc (JAMIA) 1997;4:382–93.
[27] Shiffman R, Greenes R. Improving clinical guidelines with logic and decision-table techniques. Med Decision Making 1994;14:245–54.
[28] Shwe M, Tu S, Fagan L. Validating the knowledge base of a therapy planing system. Methods Inform Med 1989;28:36–50.
[29] Tierney W, Overhage J, Takesue B, Harris L, Murray M, Vargo D, McDonald C. Computerizing guidelines to improve care and patient outcomes: the example of heart failure. J Am Med Inform Assoc (JAMIA) 1995;2:316–22.
[30] Tu S, Kemper C, Lane N, Carlson R, Musen M. A methodology for determining patients' eligibility for clinical trials. Methods Inform Med 1993;32:317–25.
[31] Tu S, Musen M. The EON model of intervention protocols and guidelines. In: Proceedings of the AMIA Annual Fall Symposium (Formerly SCAMC). Washington (DC), 1996. p. 587–91.
[32] Uckun S. Instantiating and monitoring skeletal treatment plans. Methods Inform Med 1996;35:324–33.
[33] Van der Lei J, Musen M. A model for critiquing based on automated medical records. Comput Biomed Res 1994;24(2):344–78.
[34] van Harmelen F. Applying rule-base anomalies to KADS inference structures. Decision Support Syst 1998;21(4):271–80.