in: The Workshop on Validation & Verification of Knowledge-Based Systems (V&V'98), in conjunction with the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Tento, Italy, June 1, 1998.

Multi-Level Verification of Clinical Protocols

Georg Duftschmid [1], Silvia Miksch [2], Yuval Shahar [3] and Peter Johnson [4]

[1] University of Vienna, Department of Medical Computer Sciences,
Spitalgasse 23, A-1090 Vienna, Austria,
Email: georg.duftschmid@akh-wien.ac.at

[2] Vienna University of Technology, Institute of Software Technology (IFS),
Resselgasse 3/188, A-1040 Vienna, Austria,
Email: silvia@ifs.tuwien.ac.at

[3] Stanford University, Section on Medical Informatics,
Medical School Office Building, Stanford, CA 94 305 - 5479, USA,
Email: shahar@smi.stanford.edu

[4] The Sowerby Center for Primary Health Care Informatics,
University of Newcastle, Newcastle upon Tyne, NE4 6BE, UK
Email: pete@mimir.demon.co.uk

Abstract. In the medical domain, clinical practice guidelines and protocols build a commonly accepted way to improve patient health care. During the last years various approaches have been presented to support the computerization of such guidelines and protocols. However, the verification of clinical protocols has not become an extensive research topic yet. In this paper we will present a partial and domain-specific verification approach to identify particular anomalies in protocols. Our approach is oriented on a plan-representation language of clinical protocols given as temporal, skeletal plans, called Asbru. Asbru provides the necessary hierarchical structure and task-specific knowledge roles, which are needed to divide the whole verification process in subtasks. Our verification approach examines three levels of a plan: the plan itself, all its knowledge roles and all its subplans. The final aim is to arrive at *legal* or *meaningful* plans, instead of complete or totally correct plans.

1 INTRODUCTION AND MOTIVATION

The rationalization and optimization efforts, which gained more and more importance in most market sectors during the last years, have not excluded the medical domain: medical organizations have been confronted with the fact that they are urged to increase productivity and simultaneously reduce costs without adversely affecting the quality of patient care.

One step towards this aim is the employment of commonly accepted, standardized health care procedures. Such treatment procedures are called *clinical practice guidelines* and *protocols*. In 1990, the Institute of Medicine (IOM) defined *practice guidelines* as "*systematically developed statements to assist practitioner and patient decisions about appropriate health care for specific clinical circumstances*" [Field & Lohr, 1990]. A clinical protocol is a more detailed version of a clinical practice guideline and refers to a class of therapeutic interventions.

Protocols are used for utilization review, for improving quality assurance, for reducing variation in clinical practice, for guiding data collection, for better interpretation and management of the patient's status, for activating alerts and reminders, and for improving decision support [Pattison-Gordon et al., 1996].

1.1 Automation of clinical protocols: current approaches

The computerization of protocol-based care has been the goal of various efforts within the last years. A strategy chosen by several groups is the so called *prescriptive* approach, where the system provides active interpretation of the protocols given. Examples are ONCOCIN [Tu et al., 1989] in the oncology domain, T-HELPER [Musen et al., 1992] in the AIDS domain, as well as DILEMMA [Herbert et al., 1995], EON [Tu & Musen, 1996] and the European PRESTIGE Health-Telematics project as general architectures.

The *critiquing* approach on the other hand follows a different strategy: Here the system critiques the physician's plan rather than recommending a complete one of its own. A task-specific architecture implementing the critiquing process has been generalized in the HyperCritic system [Van der Lei & Musen, 1994].

There have been efforts to use the Arden Syntax in combination with intermediate states in order to support clinical protocols [Sherman et al., 1995]. In the GEODE-CM model [Stoufflet et al., 1995] a protocol is implemented as a set of actions that are associated with the nodes of a finite-state machine. In the SPIN-approach [Uckun, 1994], protocols are represented as hierarchical skeletal plans. In [Quaglini et al., 1997] and [Fox et al., 1997] generic frameworks are suggested to support implementation of clinical protocols. Finally, several approaches are based on the hypertext browsing of protocols via the World Wide Web [Barnes & Barnett, 1995; Liem et al., 1995].

None of the current protocol-based-care systems have a sharable representation of protocols that has knowledge roles specific to the protocol-based-care tasks and is machine and human readable.

1.2 Verification of clinical protocols: why and what kind do we need?

Our aim is to support the design of clinical protocols. Ensuring reliability and enhancing quality of protocols are critical factors for their successful use in real-world applications. One way to achieve this goal is to provide an efficient verification and validation mechanism. Verification is often referred to as "building the system right", whereas validation is seen as "building the right system" [O'Keefe et al., 1987].

[Laurent, 1992] proposed a more precise definition: The verification is the sum of all processes, which attempt to determine whether a knowledge-based system (KBS) does or does not satisfy its *purely formal* specifications. Validation is the sum of verification and evaluation, the latter being the sum of all processes, which attempt to determine whether a KBS does or does not satisfy its *pseudo-formal* specifications. We are concerned with specifications derived from formalizable concepts. Accordingly, the focus of this paper is to present an approach for the verification of clinical protocols.

An essential step towards successful verification of clinical protocols is the extraction and formulation of the underlying knowledge structure. Conventional protocols are mostly expressed in *free text*, a representation that can not easily be verified. Additionally, they implicitly assume certain context, which must be made explicit. Clinical protocols embody different kinds of errors caused by the complex structure of protocols. Most protocols are partly vague and incomplete concerning their intentions and their temporal, context-dependent representation. The variability of clinical protocols (a medical goal can be achieved by different therapeutic actions) presents an additional challenge.

Considering all these shortcomings of traditional clinical protocols, we formulate the requirements for a protocol-specification language: It needs to be expressive with respect to temporal annotations and needs to have a rich set of sequential, concurrent, and cyclical operators. The language, however, also requires well-defined semantics for both the prescribed actions and the task-specific annotations, such as the plan's intentions and effects, and the preferences underlying them. These task-specific knowledge roles also facilitate protocol acquisition and verification.

Within the **Asgaard** project [Shahar et al., 1998], a temporal, intention-based, and sharable language called **Asbru** [Miksch et al., 1997] has been developed, which incorporates all of the above requirements. One of the Asgaard tasks is the verification of protocols, coded in Asbru.

Approaches concerning the verification of clinical protocols are still rather rare: [Shiffman & Greenes, 1994] are using logical analysis and application of decision-table techniques to verify and simplify clinical practice guidelines. Verification here is limited to two issues: (1) completeness and (2) unambiguousness of a guideline. A guideline is considered incomplete, if it does not provide an action for each possible value-combination of parameters, used within the guideline. Whether this assumption is realistic remains questionable: First, it seems quite probable, that some value-combinations may simply not be relevant in a certain context and therefore do not require a corresponding action. Second, this approach presumes a small number of possible discrete value-combinations, and can therefore not be used in case of parameters having infinite value domains. A guideline is considered ambiguous, if it prescribes different actions for identical value-combination of parameters. This assumption is not completely convincing either: As a simple counter-example one might imagine a guideline, which allows several, alternative therapeutic treatments to reach a certain goal. All these actions had identical activation conditions, and would such be considered ambiguous, according to [Shiffman & Greenes, 1994]. Another example that contradicts the above assumption would be a guideline, which recommends several simultaneous and complementary treatments (having identical activation conditions) for a certain situation.

[Quaglini et al., 1997] described how a guideline may be examined for logical correctness. In their approach completeness and coherency are considered as relevant proofs. Whereas completeness is defined equivalently to [Shiffman & Greenes, 1994], coherency requires compatible activation conditions for a set of conjunctive tasks (*and*-relation): A guideline is composed of hierarchical subtasks, which may be connected with an *and*-relation. Checking a guideline for coherency means to look for conjunctive sub-tasks, which exclude each other because of incompatible activation conditions. Whereas the completeness checking again is lacking a practical foundation (see arguments above), the coherency checking proposed in [Quaglini et al., 1997] seems to be useful. Several of our specifications, presented in Section 3.1.1.3 are based on the same concept of mutual exclusion.

We analyzed the applicability and usefulness of general verification methods in Software Engineering [Adrion et al., 1982] and of KBSs [Preece et al., 1992; Preece & Shinghal, 1994; van Harmelen & ten Teije, 1997] to check the correctness of clinical protocols. Our usability study showed that although some available techniques may be helpful, they are not sufficient to verify clinical protocols: The main reason is that clinical protocols incorporate several domain-specific properties, which are essential for our verification processes (see Section 3.1.1). These features are understandably not part of the verification methods, described in [Adrion et al., 1982; Preece et al., 1992; Preece & Shinghal, 1994]. Close to our approach is the task-specific validation and verification approach proposed by [van Harmelen & ten Teije, 1997] and the compositional verification designed by [Cornelissen et al., 1997]. However, their work aims towards diagnosis and we are concentrating on the verification of plans with temporal embedded actions and states. [Cornelissen et al., 1997]'s approach also covers dynamic properties of diagnosis.

We therefore propose to make use of the general verification methods (e.g. [Preece et al., 1992]) for the non-domain-specific parts of clinical protocols, and to design additional verification methods for the domain-specific parts.

Section 2 gives a brief overview about the Asgaard project and the various components of the temporal and intention-based language Asbru, which builds the basis of our verification work. Section 3 describes the verification methods we propose. In Section 4 we finally name the benefits and limitations of our approach.

2 THE ASGAARD/ASBRU PROJECT

In the Asgaard project [Shahar et al., 1998] a set of tasks and computational models are investigated, which support the execution of clinical guidelines and protocols by a care provider other than the designer of the guideline.

Within the Asgaard project, a temporal, skeletal plan-specification language, called Asbru [Miksch et al., 1997] has been developed to uniformly represent clinical protocols. Clinical protocols coded in the Asbru language are then organized within a *plan-specification library*. In the following the term *plan* is used to designate a clinical protocol, coded in the Asbru language.

Essential tasks within the Asgaard project are: (1) plan validation, including (2) plan verification during design time of a plan. During execution time the relevant tasks are (1) checking the applicability of a plan to a particular state of the world, (2) guidance in proper execution of that plan, (3) monitoring of the execution process, (4) assessment of the results of the plan, (5) critiquing the execution process and its results, and (6) assistance in modifying the original plan.

2.1 Components of Asbru

A plan consists of a name, a set of arguments, including a time annotation (representing the temporal scope of a plan), and five components: **preferences**, **intentions**, **conditions**, **effects**, and a **plan body**, which describes the actions to be executed. The general arguments, the time annotation, and all components are optional. Intentions, world states, actions/plans, and effects are durative. Therefore, we need plan states and their transition criteria to cope with the time-oriented environment. In this paper we will concentrate on the component "condition".

2.2 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. Figure 1 illustrates the different plan states and their corresponding conditions shown above the arrows. We distinguish between plan states during the plan-selection phase (left-hand side of Fig. 1) and between plan states during the execution phase (right-hand side of Fig. 1).

(1) *Filter-preconditions* need to hold initially if the plan is applicable, but can not be achieved. They are necessary for a plan to become `possible`;

(2) *Setup-preconditions* need to be achieved to enable a plan to start and allow a transition from a `possible` plan to a `ready` plan;

(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 – certain conditions (*protection intervals)* need to hold;

(5) *Abort-conditions* determine when an `activated` or `suspended` plan has to be aborted;

(6) *Complete-conditions* determine when an `activated` plan has to be completed successfully;

(7) *Reactivate-conditions* determine when a `suspended` plan has to be continued.

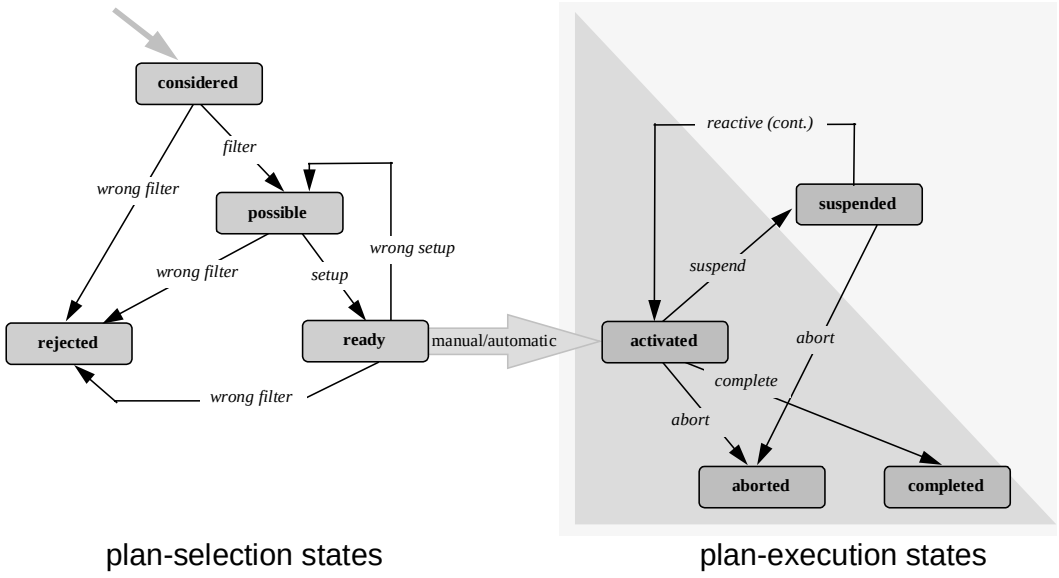

Figure 1. Plan-state model: plan states and state-transition criteria.

2.3 Decomposition of plans

Each *plan* in the plan-specification library may be composed hierarchically of a set of subplans. The execution interpreter always attempts a decomposition of a plan into its subplans, unless the plan is not found in the plan-specification library, thus representing a nondecomposable plan (informally, an *action*). This can be viewed as a "*semantic*" *stop-condition*. Such a plan is handed over to the agent for execution.

2.4 Propagation of plan states

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: the parent plan propagates its plan states to its children; a child plan propagates its plan state to its parent plan only if it is *relevant* for the parent's completion. A plan can be made relevant either implicitely or explicitely. In case of a sequential execution of subplans, all subplans have to complete as a prerequisite of their parent plan's completion and are such implicitely relevant. In case of a parallel execution of subplans, a subplan can be made relevant by explicitely naming it in the `continuation-conditions`, which are part of the Asbru language.

3 MULTI-LEVEL PLAN VERIFICATION

Our usability study of the existing verification techniques indicated that they are not sufficient for the analysis of clinical protocols (see Section 1). Therefore, we designed and developed verification methods according to our Asbru plan-representation language (see Section 2). Hereby, we concentrated on a partial, domain-specific approach, which can be seen as an extension of existing verification work. The required domain-specific knowledge is assumed to be available in a suitable knowledge-base component (see Section 3.1.3).

Our verification approach examines the components of every plan and all its subplans for the existence of several anomalies. Comparably to [Preece & Shinghal, 1994], anomalies are defined as symptoms of probable errors in plans. The goal is to arrive at *legal* or *meaningful* plans, instead of complete or totally correct plans. A plan is called *meaningful*, if it does not violate any of its specifications (see Section 3.1.1). In the first step we enumerate all specifications, which exist for any single Asbru component or for a combination of Asbru components. We examine under which circumstances each of them can be violated, and hereby make the following distinction according to the locality of the anomaly (Table 1 illustrates these three different levels):

- Level 1: anomaly within a single Asbru component;
- Level 2: anomaly within a single plan;
- Level 3: anomaly within a plan hierarchy.

In the following, we will illustrate the verification of conditions.

Table 1. Three different levels of the plan-verification.

Decomposition	Level of Checking: Level 1	Level 2	Level 3
Plan A			
Plan A_a			
Cond. A_1	✓		
Cond. A_2	✓	✓	
…	✓		
Cond. a_n	✓		
…			✓
Plan A_x			
Cond. X_1	✓		
Cond. X_2	✓	✓	
…	✓		
Cond. X_o	✓		
Plan B_a			
…			
…			

3.1 Verification of plan conditions

Any kind of protocol 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. Our verification method of conditions is based on the following three features of the Asbru language:

- the plan state model;
- the hierarchical structure of plans;
- the conditions to control state transitions.

These components are basic elements of protocol automation and are also partly incorporated in most other approaches ([Fox et al., 1997], [Herbert et al., 1995], [Quaglini et al., 1997], [Sherman et al., 1995], [Tu & Musen, 1996], [Uckun, 1994]). Consequently, the verification approach we propose may not only be useful for plans, expressed in the Asbru language, instead it may be applicable to a wide scope of protocol models.

In the following, we will define the specifications concerning Asbru conditions both informally and formally, discuss the relevant concepts, and show the circumstances under which the specifications can be violated. We will hereby distinguish the resulting anomalies according to their scope (compare Table 1). Finally, the order of the "3-level checking" is illustrated in pseudo-code.

3.1.1 Mapping plans to rule bases

[Preece et al., 1992] gave a detailed summary of anomalies, that can occur in rule bases. For each anomaly a strict and unambiguous definition is provided.

In the following we will show, how the Asbru plan-state model and every single plan can be transformed to a rule base. We can then refer to [Preece et al., 1992] 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 [Preece et al., 1992]. These existing anomalies have been complemented with additional ones, which result from specifics of the

Asbru language and do not have an equivalent in [Preece et al., 1992]. Within the following specifications we will indicate at each case the origin of the anomaly.

After some definitions, we will first outline the complete and generic rule base "*MPS*" for the Asbru Model of Plan States (see Figure 1). Its rules specify in which sequence the conditions of a plan are considered. The rule base *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 rule base 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 $PC = PC_1 \cup ... \cup PC_n$. In each PC_i we should have exactly one rule with a consequence, which is an element of *ConditionSet*. As the rule bases PC_i are to be implemented by the user, they will be the source of our verification methods.

3.1.1.1 Relevant 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)}

StartingStateSet = {considered(pl,pa)}

IntermediateStateSet = {possible(pl, pa), ready(pl, pa), activated(pl, pa), suspended(pl, pa)}

FinalStateSet = {rejected(pl, pa), aborted(pl, pa), completed(pl, pa)}

StateSet = StartingStateSet ∪ IntermediateStateSet ∪ FinalStateSet

Note that the *ConditionSet* correlates to the arrows in Figure 1 and the *StateSet* correlates to the boxes in Figure 1.

Rule $R = L_1 \wedge ... \wedge L_n \rightarrow M$
$antec(R) = \{L_1, ..., L_n\}$
$conseq(R) = M$

SS_i = Set of all subplans of plan *i*

RS_i = 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 their parent plan, either explicitely or implicitely.

A hypotheses *H* is *inferable* from a rule base *RB* if there is some environment *E* such that *H* is a logical consequence of supplying *E* as input to *RB*. In our case *E* is represented by a patient, incorporating certain characteristics. The patient's characteristics may be queried by means of suitable predicates, such as *female(pa)* or *insulin-indicator(pa)*.

Formally*: inferable(H,RB,E)*
iff $(RB \cup E) \vdash H$, $E \in PatientSet$

A rule $R \in RB$ is *fireable* if there is some environment *E* such that the antecedent of *R* is a logical consequence of supplying *E* as input to *RB*.

Formally: *fireable(R,RB,E)*
iff $(\exists Substitution\ \sigma)\ (RB \cup E) \vdash antec(R)\sigma$,
$E \in PatientSet$

3.1.1.2 Rule Bases *MPS* and *PC*

Table 2 shows the rule base 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 Figure 1).

Table 2. Rule Base *MPS*: 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)*

A simplified version of a certain PC_k, a rule base for all conditions of the plan to treat noninsulin-dependent gestational diabetes mellitus (*"GDM-TYPE II")* may for example look like:

Table 3. Rule Base PC_k: conditions of plan *GDM-TYPE II*.

female(pa) $\wedge$ *pregnant(pa)* $\rightarrow$ *filter(GDM-TYPE II, pa)*
test_available(glucose-toler., pa) $\rightarrow$ *setup(GDM-TYPE II, pa)*
activate(GDM-TYPE II, pa) /* automatic activation */
delivered(pa) $\rightarrow$ *complete(GDM-TYPE II, pa)*
state(blood-glucose, high, pa) $\rightarrow$ *suspend(GDM-TYPE II, pa)*
state(blood-glucose, normal, pa) $\rightarrow$ *reactivate(GDM-TYPE II, pa)*
insulin-indicator(pa) $\rightarrow$ *abort(GDM-TYPE II, pa)*

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 rule base for the Asbru plan state model, formally $MPS \cup PC_i$.

To handle the anomalies, which may occur within a rule base $MPS \cup PC_i$, it is first necessary to stress the specific properties of this rule base, which distinguish it from a "general" rule base:

- The set of possible consequences of rules we have to consider is very small: As we mentioned before, only the rule bases PC_i are modifiable by the user and for each $C \in ConditionSet$ there should exist exactly one rule *R* in each PC_i with $conseq(R) = C$. Hence, each PC_i should contain at most seven rules, each with a different consequence. Evidently, it will be possible to enforce this demand without much effort: This might be done by providing some sort of input support to the user or even by manual verification due to the small number of rules. As we can more or less rule out the possibility of a PC_i containing several rules with identical consequences, a major part of the anomalies presented in [Preece et al., 1992] become obsolete in our case.
- The amount of rules we have to consider when checking anomalies within one plan is limited to the scope of the plan's hierarchy at maximum: Plans of different hierarchies are completely independent, hence no anomaly may 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 base *MPS*: In contrary to [Preece et al., 1992], which refers to "general" rule bases, we can therefore include the order of inference into our considerations.

3.1.1.3 Specifications for MPS ∪ PC

Now we will list all specifications, which must hold for a plan. Each specification will first be presented informally, then a formal definition will be given.

We must consider the case that a specification can be relevant in multiple levels of plan verification (see Section 3). Therefore, the specifications in this section will be given in a general way, ignoring any affiliation to one of the three levels of plan verification. In Section 3.1.4 we will examine the anomalies within the particular levels of plan verification.

- **Every condition must have a chance to be satisfied**

Clearly, any single condition being part of a plan must have a chance to be satisfied during execution of the plan.

Returning to the knowledge base vocabulary, we equivalently specify that each rule of *PC* must be fireable. This is equivalent to [Preece et al., 1992], who treat the violation of this request as a special case of a *redundant rule*.

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

- **Every valid sequence of plan states must be reachable**

Any sequence of plan states, which defines a valid path according to the Asbru plan-state model (see Figure 1), must also statically be possible within a single plan. However, we do not demand that every plan state is actually reached during execution time. This request concerns the Asbru plan-state model and it is not discussed in [Preece et al., 1992].

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 of the same patient in PC_i, which have fired before. For each rule *R* in PC_i, the set of rules, which must fire before considering *R* is defined through *MPS*. For example, the `activate-conditions` of a plan is not considered, before its `filter-` and `setup-conditions` have been satisfied, as *R8* cannot fire in *MPS* before rules *R2* and *R4* have fired.

As we are talking about valid sequences of plan states that must be reachable, it is more intuitive to refer directly to *MPS*: The consequences of the rules in *MPS* build the set of all possible plan states, and the rules themselves define the valid sequences of plan states. Informally we can therefore specify, that for each plan *i* there must exist an environment *E*, such that each consequence in *MPS* is inferable by supplying *E* as input to $MPS \cup PC_i$.

Formally our specification will then look like:

$$(\exists E, \sigma)\ inferable(conseq(R)\sigma, MPS \cup PC_i, E)$$
$$\forall R \in MPS,\ 1 \le i \le n$$

- **Every plan must be able to complete**

Obviously, a plan will only be able to complete, if its `complete-conditions` and all preceding conditions can be satisfied. This is a specialization of the above specification and it is also not handled in [Preece et al., 1992].

Equivalently we can demand that *conseq(R10)* must be inferable in every $MPS \cup PC_i$. Nevertheless this is not sufficient: Each plan may have one or more subplans, that are executed either sequentially, concurrently, or cyclically. In case of a sequential execution, all subplans have to complete as a prerequisite for their parent's completion. In case of a parallel execution, it is possible to explicitly specify which subplans have to complete. Consequently, an additional prerequisite for a plan's completion is that the `complete-conditions` and all predecessor conditions can be satisfied for all subplans that are relevant for its completion.

Using RS_i as the set of all subplans, relevant for the completion of plan *i*, we informally specify: As mentioned above, for each plan *i* there must exist an environment *E* such that the consequence of rule $R10 \in MPS$ is inferable by supplying *E* as input to $MPS \cup PC_i$. Additionally the consequence of rule $R10 \in MPS$ must be inferable by supplying the same *E* as input to $MPS \cup PC_k$ for all $k \in RS_i$.

Formally:

$(\exists E,\sigma)\ [inferable(conseq(R10)\sigma, MPS \cup PC_i, E) \wedge$
$inferable(conseq(R10)\sigma, MPS \cup PC_k, E)]$
$\forall (1 \le i \le n,\ k \in RS_i), R10 \in MPS$

- **A subplan should not be stopped by its parent plan**

We mentioned in Section 2.4 that the final plan states {`rejected`, `aborted`, and `completed`} and the state `suspended` 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 as an error, but a warning. This request is related to specific plan states of the Asbru plan-state model and has no equivalent in [Preece et al., 1992].

In terms of our knowledge base we informally specify:

SS_i = set of all subplans of plan *i*

$FSR = \{R3, R5, R6, R9, R10, R13\} \subseteq MPS$: set of all rules with a consequence $\in$ *FinalStateSet*

$R11 \in MPS$: rule with consequence *suspended(pl, pa)*

Then we demand for each plan *i*: For each rule *R* in *FSR* the base PC_i must ensure that whenever *R* fires, there is also a rule *R'* in *FSR* that fires, for each of plan *i*'s subplans.

Equivalently, we specify for the `suspend-conditions`: Whenever *R11* fires for plan *i*, rule *R11* must also fire for each of plan *i*'s subplans.

Formally:

$(\exists\ \sigma)\ antec(R) \rightarrow antec(R')\sigma,$
$\forall (R \in FSR \cup PC_i,\ i = 1..n, R' \in FSR \cup PC_k,\ k \in SS_i)$

$(\exists\ \sigma)\ antec(R11) \rightarrow antec(R11')\sigma,$
$\forall (R11 \cup PC_i,\ i = 1..n, R11' \in R11 \cup PC_k,\ k \in SS_i)$

- **No state of a plan should be skipped**

If the condition of a plan state is unconditionally true as soon as it is checked, because of one or more poorly designed conditions, the plan state at which the condition is checked is "skipped" and thus unnecessary. Hence our first request will be that there must not be any single condition, which is inherently true.

Accordingly, we specify informally that there must not be any rule in *PC*, which is fireable for every possible environment *E*. In particular, PC must not contain facts, except for the case of an automatic `activation-conditions`. As this is not considered an anomaly in general, it is clearly not referred to in [Preece et al., 1992].

Formally:

$\neg ((\exists R \in PC)\ fireable(R,PC,E)\ (\forall E \in PatientSet))$

Our second request concerns the order in which conditions are considered, enforced by *MPS* (see above). We demand for every pair of conditions within the same plan: The firing of the preceding condition in the inference chain, induced by *MPS*, must not entail the firing of a condition, considered later in the inference chain. Instead every condition in the chain should check additional information in order to make sense.

Formally:

$\neg ((\exists\ \sigma)\ antec(R) \rightarrow antec(R')\sigma,$
$\forall (R, R' \in MPS \cup PC_i,\ i = 1..n, R \prec R'))$
$R \prec R'$ means R is considered first in the inference chain.

This request somehow resembles the *subsumed rule* anomaly in [Preece et al., 1992]. However it differs from [Preece et al., 1992] in the following two points: First, we do not presume identical consequents for R and R'. Second, the sequence of R and R' within the inference chain is relevant for us.

- **No plan must loop eternally**

There must not be any plan in the library, which allows looping within its states. Otherwise a *circularity* would be present, according to [Preece et al., 1992]. Looking at the Asbru plan state model, the most obvious potential loops may occur between states `possible` $\leftrightarrow$ `ready`, and `activated` $\leftrightarrow$ `suspended`. Looping between `possible` and `ready` is not possible as conditions `setup` and $\neg$`setup` cannot be satisfied concurrently. Anyway, looping between `activated` and `suspended` is possible, if the `suspend-` and `reactivate-conditions` might be satisfied at the same time.

Therefore, we informally specify that the firing of *R11* in $MPS \cup PC_i$ must not infer the firing of *R12* in $MPS \cup PC_i$ for any plan *i*. This can be reduced to the demand that for any PC_i, the firing of the rule having *suspend(pl,pa)* as its consequence, must not entail the firing of the rule having *reactivate(pl,pa)* as its consequence. Actually this is a special case of the request that no plan state must be skipped.

Formally:

$\neg ((\forall R, R' \in PC_i)\ (conseq(R) = suspend(plan\ i,pa)) \wedge$
$(conseq(R') = reactivate(plan\ i,pa)) \wedge (antec(R) \rightarrow$
$antec(R')\sigma)\ (\forall i = 1..n))$

3.1.2 Relevant concepts

The following concepts are relevant in order to examine whether a plan complies to the above specifications.

Semantic constraint expression: This is an expression $\{L_1, ..., L_n\}$, which is interpreted as meaning that the simultaneous truth of $L_1 \wedge ... \wedge L_n$ would not make semantic sense. For example, the set *{male(x), pregnant(x)}* says that, for all *x*, *x* cannot be both a male and pregnant. This concept is equivalent to the *impermissible set* or *semantic constraint expression*, used in [Preece et al., 1992], [Preece & Shinghal, 1994].

ConstraintSet = Set of all semantic constraint expressions

Mutual exclusiveness: *Two conditions corresponding to rules R_i and $R_j \in PC$ are mutually exclusive, if their simultaneous firing infers a semantic constraint expression $C \in ConstraintSet$.*

Note that in our case mutual exclusiveness of conditions can only result from incompatibility between the antecedents of their corresponding rules: As mentioned before, the consequences of all rules in *PC* originate from *ConditionSet*, and there is no semantic constraint expression defined in *ConstraintSet* concerning elements of *ConditionSet*.

Entailment: *A condition corresponding to rule $R_i \in PC$ entails another condition corresponding to rule $R_j \in PC$, if $R_i \rightarrow R_j$.*

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).

3.1.3 Domain-specific assumptions

We assume that the verification methods have access to a domain-specific knowledge-base, which contains the types and domains of all condition parameters. It also includes the set of all semantic constraint expressions, and can such provide information about the mutual exclusiveness of conditions. Finally, it allows to query entailments between conditions.

3.1.4 Anomalies

Level 1: Within a single condition

- **Every condition must have a chance to be satisfied**

This specification can for example be violated in case of medically implausible conditions (e.g. domain or type violations). Another possibility would be a condition defined by a rule, the antecedent of which contains a semantic constraint expression.

Level 2: Within a single plan

- **Every valid sequence of plan states must be reachable**

If any two conditions, which belong to a plan-state pair of a valid path, are mutually exclusive, the plan-state of the condition checked later could never be satisfied and thus a plan-state transition is never reachable. The possible mutually exclusive parameters are domain-specific and are defined in the knowledge-base component (compare Section 3.1.3)

Example:

male(pa) → filter(PlanX, pa)
pregnant(pa) → setup(PlanX, pa)

Explanation: plan state `ready` can never be reached.

- **Every plan must be able to complete**

This is a specialization of the above anomaly: If the `complete-conditions` and any other condition of a plan within the same path of plan states, are mutually exclusive, the plan will not be able to complete.

Example:

blood-group-A(pa) → filter(PlanX, pa)
blood-group-B(pa) → complete(PlanX, pa)

Explanation: plan state `complete` can never be reached.

- **No state of a plan should be skipped**

This constraint may be violated, if two conditions are defined in a way, that condition checked first entails the condition checked second. The plan state at which the second condition is checked, would be "skipped" in any case.

Example:

older(pa, 20) → filter(PlanX, pa)
older(pa, 15) → setup(PlanX, pa)

Explanation: plan state `possible` would be skipped.

- **No plan must loop eternally**

A plan might loop eternally, if its `suspend-conditions` entails its `reactivate-conditions`.

Example:

value_higher(pa, cholesterol, 250)
→ suspend (PlanX, pa)
value_higher(pa, cholesterol, 200)
→ reactivate (PlanX, pa)

Explanation: *PlanX* would loop for each patient with a cholesterol-level higher 250.

Level 3: Between several plans of a hierarchy

- **Every plan must be able to complete**

In order to complete, a plan must pass the `filter-`, `setup-` and `complete-conditions` (we omit the `activate-condition` here, as it does not refer to patient data). Plan A will not complete, if there is a mutually exclusive pair of the above conditions within one plan relevant for plan A's completion, or within any two plans relevant for plan A's completion (see Figure 2).

Example:

/ SubplanAa is a relevant subplan of PlanA */*

blood-group(pa, A) → filter(PlanA, pa)
blood-group(pa, B) → setup(SubplanAa, pa)

Explanation: *PlanA* will not be able to complete.

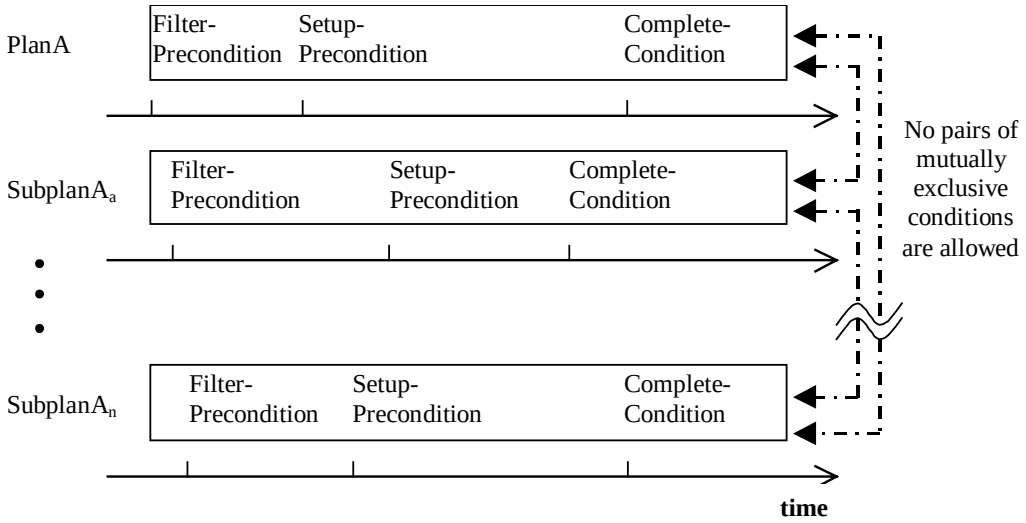

Figure 2. Example of a plan with its relevant subplans and the corresponding pairs of mutually exclusive conditions.

- **A subplan should not be stopped by its parent plan**

The only way to ensure that this specification holds is, that each plan's `reject-`, `abort-` and `complete-conditions` entails one of these conditions for all its subplans. The same must hold for each plan's `suspend-condition`.

Example:

/ PlanA has subplans SubplanAa and SubplanAb */*

$value\_higher(pa, bilirubin, 5) \wedge$
$value\_higher(pa, GOT, 22) \rightarrow complete(PlanA, pa)$
$value\_higher(pa, bilirubin, 5)$
$\rightarrow complete(SubplanAa, pa)$
$value\_higher(pa, GOT, 22)$
$\rightarrow complete(SubplanAb, pa)$

Explanation: *PlanA* will not stop its subplans.

3.2.5 Pseudo-code "3-level checking" of conditions

The method "checkConditions" at class "Plan" could look like:

```
/*"self" refers to receiving plan instance */
checkConditions                  /* method-name */
  self subplans iterate:
[subplan checkConditions]./* recursive call */
  self conditions iterate:
    [condition checkLocally].    /* level 1 */
  self checkLocally.             /* level 2 */
  self checkHierarchy.           /* level 3 */
```

Note that it will usually not be necessary to recheck all 3 levels, in case of a modification of a meaningful plan hierarchy: if a new plan is added to a meaningful plan hierarchy for example, level 1 and 2 checks have to be made only for the new plan, whereas the complete hierarchy can be rechecked with one level 3 check.

4 CONCLUSIONS

In this paper we have presented a domain-specific, partial verification approach to identify violations of particular anomalies, which may occur in clinical protocols. The outcome of our verification process is to arrive at *legal* or *meaningful* plans, instead of complete or totally correct plans. The approach is used to check the correctness of protocols, coded in the temporal, skeletal plan-specification language Asbru. We utilized three anomalies defined by [Preece et al., 1992]. Two of them were directly usable (namely, circular dependency and redundant rule), the third one (subsumed rule) was adapted for our specific plan representation. Additionally, we enhanced [Preece et al., 1992]'s approach, applying our hierarchical structure and our particular knowledge roles.

The advantages of our verification approach are that (1) partial checks are not computationally expensive; (2) domain-specific knowledge is considered; (3) incremental verification can be used to further reduce computational efforts: Moving a meaningful plan to a new hierarchy only requires level 3 checking, for example; (4) we utilize elements, which are essential for clinical protocols in general and therefore our verification approach should be applicable to a wide scope of protocol representations.

The limitations are that (1) partial checks do not guarantee a completely correct plan, (2) we ignored dynamic properties.

[van Harmelen, 1998] showed how the usefulness of the anomalies described in [Preece et al., 1992] and [Preece et al., 1994] can be improved by reformulating them in terms of conceptual models, in particular KADS inference structures. This allows applying the anomalies to a much wider class of KBS. Whether this approach can also be helpful in our case of clinical protocol verification, will be a subject of our future work.

REFERENCES

W. R. Adrion, M. A. Branstad, and J. C. Cherniavsky, 'Validation, Verification, and Testing of Computer Software', *Computing Review*, **14**(2), 159-192, (1982).

M. Barnes, G. O. Barnett, 'An Architecture for a Distributed Guideline Server', in *Proceedings of the Annual Symposium on Computer Applications in Medical Care (SCAMC-95)*, New Orleans, Louisiana, 233-7, (1995).

F. Cornelissen, C. M. Jonker, J. Treur, 'Compositional Verification of Knowledge-Based Systems: a Case Study for Diagnostic Reasoning', in *Proceedings of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV'97)*, (1997).

M. J. Field, K. H. Lohr (eds), *'Clinical Practice Guidelines: Directions for a New Program*', Institute of Medicine, Washington DC: National Academy Press, (1990).

J. Fox, N. Johns, and A. Rahmanzadeh, 'Protocols for Medical Procedures and Therapies: A Provisional Description of the PROforma Language and Tools', in *Artificial Intelligence in Medicine, 6th Conference on Artificial Intelligence in Medicine Europe (AIME-97)*, Grenoble, France, March 23-26, 21-38, (1997).

S. I. Herbert, C. J. Gordon, A. Jackson-Smale, and J.-L. Renaud Salis, 'Protocols for Clinical Care', *Computer Methods and Programs in Biomedicine*, **48**, 21-26, (1995).

J.-P. Laurent, 'Proposals for a Valid Terminology in KBS Validation', in *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI'92)*, 829-834, (1992).

E. B. Liem, J. S. Obeid, P. E. Shareck, L. Sato, R. A. Greenes, '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, Louisiana, 223-7, (1995).

S. Miksch, Y. Shahar, and P. Johnson, 'Asbru: A Task-Specific, Intention-Based, and Time-Oriented Language for Representing Skeletal Plans', in *7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)*, Milton Keynes, UK, (1997).

M. A. Musen, C. W. Carlson, L. M. Fagan, S. C. Deresinskim, E. H. Shortliffe, 'T-HELPER: Automated Support for Community-Based Clinical Research', in *Proceedings of the 16th Annual Symposium on Computer Applications in Medical Care (SCAMC-92)*, 719-23, (1992).

R. M. O'Keefe, O. Balci and E. P. Smith, 'Validating Expert System Performance', in *IEEE Expert*, **2**(4), 81-90, (1987).

E. Pattison-Gordon, J. J. Cimino, G. Hripcsak, S. W. Tu, J. H. Gennari, N. L. Jain, and R. A. Greenes, *'Requirements of a Sharable Guideline Representation for Computer Applications*', Stanford University, Report No. SMI-96-0628, (1996).

A Preece, R Shinghal, and A Batarekh. 'Principles and Practice in Verifying Rule-Based Systems', in *Knowledge Engineering Review*, **7**(2), 115-141, (1992).

A. Preece and R. Shinghal, 'Foundation and Application of Knowledge Base Verification', in *International Journal of Intelligent Systems*, **9**(8), 683-702, (1994).

S. Quaglini, R. Saracco, M. Stefanelli, and C. Fassino, 'Supporting Tools for Guideline Development and Dissemination', in *Artificial Intelligence in Medicine, 6th Conference on Artificial Intelligence in Medicine Europe (AIME-97)*, Grenoble, France, 39-50, (1997).

Y. Shahar, S. Miksch, and P. Johnson, 'The Asgaard Project: A Task-Specific Framework for the Application and Critiquing of Time-Oriented Clinical Guidelines', *Artificial Intelligence in Medicine*, to appear, (1998).

E. H. Sherman, G. Hripcsak, J. Starren, R. A. Jender, and P. Clayton, 'Using Intermediate States to Improve the Ability of the Arden Syntax to Implement Care Plans and Reuse Knowledge', in *Annual Symposium on Computer Applications in Medical Care (SCAMC-95)*, New Orleans, Louisiana, 238-242, (1995).

R. N. Shiffman, and R. A. Greenes, 'Improving Clinical Guidelines with Logic and Decision-Table Techniques', in *Medical Decision Making 1994*, 245-254, (1994).

P. E. Stoufflet, S. R. A. Deibel, J. H. Traum, R. A. Greenes 'A State-Transition Method of Modeling Clinical Encounters', *Proceedings of the AMIA Spring Congress 1995*, (1995)

S. W. Tu and M. A. Musen, 'The EON Model of Intervention Protocols and Guidelines', in *1996 AMIA Annual Fall Symposium (formerly SCAMC)*, Washington, DC., 587-591, (1996).

S. W. Tu, M. G. Kahn, M. A. Musen, J. C. Ferguson, E. H. Shortliffe, L. M. Fagan 'Episodic Skeletal-Plan Refinement on Temporal Data', in *Communications of the ACM, 32* , 1439-55, (1989).

S. Uckun, *Instantiating and Monitoring Skeletal Treatment Plans*, Knowledge Systems Laboratory, Stanford University, Stanford, CA 94305, USA KSL 94-49, (1994).

J. Van der Lei, M. A. Musen, 'A Model for Critiquing based on Automated Medical Records', *Computers and Biomedical Research*, **24**(2) , 344-78, (1994).

F. van Harmelen and A. ten Teije, 'Validation and Verification of Conceptual Models of Diagnosis', in *Proceedings of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems (EUROVAV'97)*, 117-128, (1997).

F. van Harmelen, 'Applying Rule-Base Anomalies to KADS Inference Structures', in *Decision Support Systems*, 21(4), 271-280, (1998).