

Verification of temporal scheduling constraints in clinical practice guidelines

Georg Duftschmid¹⁾, Silvia Miksch²⁾, Walter Gall¹⁾

1) University of Vienna, Department of Medical Computer Sciences
Spitalgasse 23, A-1090 Vienna, Austria
{georg.duftschmid,walter.gall}@akh-wien.ac.at, www.akh-wien.ac.at/imc/

2) Vienna University of Technology, Institute of Software Technology
Favoritenstraße 9-11/188, A-1040 Vienna, Austria
silvia@ifs.tuwien.ac.at, www.ifs.tuwien.ac.at/

Address requests for reprints and correspondence to:

Georg Duftschmid

University of Vienna, Department of Medical Computer Sciences,
Spitalgasse 23, A-1090 Vienna, Austria,
Tel.: +43-1-40400 / 6696
Fax.: +43-1-40400 / 6697

Email: georg.duftschmid@akh-wien.ac.at

Keywords:

Clinical practice guidelines, verification, temporal scheduling constraints, medical plan management

sub/a AIM-J; November 2001

Abstract

The computerization of clinical practice guidelines presents an important scientific challenge to the medical informatics community. One factor complicating this effort, which has been frequently reported in literature, is the existence of deficiencies within guideline knowledge: As safety is a critical issue in the medical domain, guidelines that might lead to wrong patient treatment due to unclear or even wrong instructions are not acceptable for implementation. In this paper, we focus on the detection of flaws within temporal scheduling constraints, a type of knowledge that has been incorporated by several current guideline representation formats. For this purpose, a suitable verification approach is presented, which is based on well-established temporal constraint satisfaction techniques, and serves two additional purposes: (1) It yields suggestions for an equivalent, but more explicit representation of non-minimal constraints, (2) It can be used by the guideline interpreter to assemble feasible execution intervals for each guideline activity. We will refer to the guideline-representation language *Asbru* as the demonstration medium of our approach, as it provides the most powerful functionality in temporally constraining guideline execution in comparison with other current formats. Our approach is reusable for the verification of several alternative guideline-representation formats.

1 Introduction

The computerization of clinical practice guidelines and the different tasks associated with this goal have been the subject of numerous research projects in the medical informatics community throughout the last decade [4, 7, 12, 15, 16, 22]. In comparison with the extensive research efforts invested in this subject, few guidelines have actually been employed in an automated version in clinical practice. One factor complicating the implementation of such applications, which has frequently been reported in literature, is the existence of deficiencies within guideline knowledge [10, 14, 28]: As safety and transparency are critical issues in the medical domain, guidelines that might lead to wrong patient treatment due to unclear or even wrong instructions are not acceptable for implementation.

In Software Engineering in general and in Knowledge-Based Systems in particular, a common strategy to ensure the correctness of a system consists in its verification. As clinical guidelines encode a multitude of different knowledge types (e.g., *goals*, *conditions*, *effects*), which might all be used incorrectly, the application field for verification is broad in this domain. One such type of knowledge, that builds a relevant subject for verification, is the concept of *temporal scheduling constraints* within a guideline. The focus of this paper will be to present an approach, that enables the verification of guidelines during their design phase by checking whether the following three types of scheduling constraints are consistently applied:

1. Temporal scheduling constraints
2. Scheduling constraints implied by the guideline's control flow
3. Scheduling constraints implied by a hierarchical structuring of a guideline

The presented approach is based on the calculation of the minimal, temporal network, and serves three purposes: (1) It allows the identification of inconsistent temporal scheduling constraints, (2) It examines temporal scheduling constraints for their minimality and yields the most explicit, equivalent representation as a suggestion for the adaptation of non-minimal temporal scheduling constraints, (3) It can be used by the guideline interpreter to determine feasible execution intervals for each guideline activity.

The computation of the minimal network can be done in $O(n^3)$ time for a guideline with n individual activities and therefore constitutes an efficient method for its verification. Efficiency is an issue for us insofar as it allows our verification process to be applied interactively during the design phase of a guideline, e.g. after each definition of a new temporal scheduling constraint, without annoying the user by long answering times.

As we will show, scheduling constraints of the above three types have been integrated to a noticeable extent in most current guideline representation formats. We will refer to the Asbru language as the demonstration medium of our approach, as it provides the most powerful functionality in temporally constraining guideline execution. However, our work is reusable within other guideline representations to a degree that corresponds to their integration of temporal scheduling constraints.

The paper is organized as follows: In section 1.1 we will examine to which extent scheduling constraints are integrated in current guideline representation formats. After providing an overview on existing work about guideline verification in section 1.2, the plan-specification language Asbru for the representation of guidelines will be introduced in section 1.3. In section 1.4 we will show how our verification task corresponds to a simple temporal problem. The verification of scheduling constraints within a clinical guideline will be handled in section 2: The main issue of this section will be to examine the possible origin of inconsistencies and non-minimalities of temporal scheduling constraints. The method we use to verify temporal scheduling constraints will be described in section 3. It detects inconsistent and non-minimal temporal scheduling constraints and yields a suggestion for the modification of non-minimal temporal scheduling constraints. After showing some results from practically applying our approach to a fictive guideline in section 4, we will conclude the paper in section 5.

1.1 Integration of scheduling constraints in guideline representation formats

In this section, we will examine how scheduling constraints of the above three types are implemented within the most important, current guideline representation formats.

1.1.1 Temporal scheduling constraints

- In its original version, the GLIF model included temporal constraints on patient data elements only [16]. Their purpose was to define the required actuality of a data element to be considered within the guideline. However, in the current version 3 a structured grammar, based on the Arden Syntax logic grammar, has been added that allows, amongst others, the specification of temporal expressions [18]. This enhancement of its temporal expressiveness is planned to be utilized in the form of duration constraints on actions and decisions.
- PROforma models guidelines as plans, which may consist of one or more tasks [4]. Each plan may define temporal constraints on the enactment of tasks, which are specified in the form of time durations between tasks.
- The DILEMMA approach and its successor, the PRESTIGE project, allow temporal expressions to be included in conditions, which control the transitions between the different states, their guidelines may adopt [5, 7]. Therefore, it should be possible to determine a guideline's starting time point by adding temporal constraints to its eligibility criteria.

- The EON approach enables the specification of complex, temporal expressions including temporal abstraction by incorporating the RÉSUMÉ system [15]. This functionality is only used to extend EON's expressiveness when referring to patient data though. To our knowledge, it has not yet been applied to constrain the scheduling of guidelines, implemented within EON.
- Sherman and colleagues describe how guidelines may be implemented within the Arden Syntax [22]. In their approach, a guideline is represented as a set of MLMs, which are synchronized by means of so-called *intermediate states*: By storing a specific coded value in the database, an MLM triggers its successor MLMs, which define this particular event in their *evoke slot*. The Arden Syntax allows the specification of a time delay within the *evoke slot*, which builds an offset from the triggering event and has to be awaited before the MLM is actually started [8]. The *time delay* statement may consequently be used to specify minimum duration constraints between individual MLMs.
- An approach, which lies particular emphasis on the efficient handling of temporal expressions within guideline knowledge is the *Asbru* language, which was created as part of the *Asgaard* project [12, 21]. As we will demonstrate in section 1.3, Asbru provides the most powerful functionality in specifying temporal scheduling constraints in comparison with the above approaches.

1.1.2 Scheduling constraints implied by the guideline's control flow

Integration of scheduling constraints based on a guideline's control flow was already indirectly postulated by Pattison-Gordon and colleagues in their paper on requirements of a sharable guideline representation [17]. Their demand for an explicit representation of *temporal sequence* addresses the ability to express different variants of control flow within computerized guidelines, such as sequential, parallel and iterative execution of guideline actions. All of the above guideline representation techniques have implemented this functionality. We will describe in section 2.1.2 what kind of scheduling constraints may result from a guideline's control flow.

1.1.3 Scheduling constraints implied by a hierarchical structuring of a guideline

Ohno-Machado and colleagues specified a hierarchical structuring of guidelines, referred to by *decomposability of actions* by them, as a representational requirement of the GLIF model [16]. All other representation formats, except the approach based on linking MLMs by intermediate states, equally rely on a hierarchical organization of guidelines. We will describe in section 2.1.3 what kind of scheduling constraints result from the hierarchical structuring of a guideline.

1.2 Related work on the verification of guidelines

Compared with the number of presented approaches for the computerization of clinical guidelines and the different associated tasks that have been tackled, little can be found about the verification of clinical guidelines in literature yet: Shiffman and Greenes used logical analysis and application of decision-table techniques to verify and simplify clinical practice guidelines [24]. Guidelines are seen as sets of condition-action pairs and verification consists of proving their *completeness* and *unambiguousness*. A guideline is considered incomplete, if it does not provide an action for each possible value-combination of parameters, used within conditions. It is considered ambiguous, if it prescribes different actions for identical value-combinations of parameters. Shiffman refined the proposed method by showing how decision-tables, representing the guideline's logic, may be augmented by additional information, such as cost, risk, evidence or literature citations [23]. Miller and colleagues further improved Shiffman and Greenes' approach by considering only those value-

combinations, which are semantically possible [13]. Quaglini and colleagues described how a guideline may be checked for *completeness* and *coherency* [19]. Whereas completeness is defined equivalently to Shiffman and Greenes, checking a guideline for coherency means to look for conjunctive actions, which exclude each other because of incompatible activation conditions. Duftschmid and Miksch proposed a three-step method to verify conditions within a clinical guideline, modeled as a hierarchy of Asbru plans [3]. Hereby, a guideline is checked for the occurrence of anomalies, which represent violations of certain properties, postulated from a legal guideline's condition set. According to their scope, three types of anomalies are distinguished, which are anomalies *within a single condition*, *within a single plan*, and *within a hierarchy of plans*.

All above approaches concern the verification of condition-based, clinical guidelines only and do not consider temporal information. Guarnero and colleagues described an approach for the representation of clinical guidelines that allows the specification of minimal and maximal durations for individual guideline actions [6]. They use a temporal reasoning system called LaTeR for examining the consistency of these duration intervals with delay intervals, which may be defined between pairs of actions to be executed in sequence.

Other related work can be found in the domain of temporal constraint propagation, which can be distinguished according to the type of constraints reasoned about:

- Qualitative constraints: Allen's interval algebra and Vilain and Kautz's point algebra are two fundamental representatives of this direction [1, 30].
- Quantitative constraints: This class of constraints was originally studied by Dechter and colleagues [2].
- Combination of qualitative and quantitative constraints: Two important approaches that integrate both types of constraints were presented by Meiri, respective Kautz and Ladkin [9, 11].

For our problem, work on temporal reasoning with quantitative constraints and the combined variant is of particular interest: In our guideline representation language, temporal scheduling constraints are quantitative. Scheduling constraints induced by control flow and hierarchical structuring are qualitative but may easily be translated to quantitative ones. Several authors have focused on the propagation of combined, quantitative and qualitative constraints in temporal networks, supporting multiple levels of uncertainty:

Rit's approach of propagating temporal constraints in scheduling problems is close to ours as it is based on the same structure for representing temporal scheduling constraints, he calls SOPOs [20]. Here, the goal is to solve the so-called *constrained occurrences problem*, where a network of SOPOs, connected by qualitative constraints, is compressed to those occurrences, which are compatible with the constraints between them. Whereas inconsistencies are detected, the approach does not assure minimality of SOPOs. The paper further suggests a two-dimensional, graphical representation and propagation of constraints. Although this visualization allows an exact, graphical representation of an event with uncertain start, end and duration, it suffers from a lack of intuitive comprehensibility. Stillman and colleagues further extend Rit's work by allowing parameterized qualitative as well as quantitative constraints to be defined between SOPOs [27].

Vere described several constraints that must hold within a *temporal window* of an activity and presented an algorithm for the propagation of constraints between several windows of sequential and consecutive activities [29]. However, his temporal windows are simpler than our temporal scheduling constraints, as he assumes a fixed duration and does not support uncertainty in their finishing time point.

Our main point of orientation was the work on *simple temporal problems* presented by Dechter and colleagues [2]. Their approach is very efficient and covers all of our requirements, except for unordered sequential execution of guideline activities. This type of control flow cannot be tackled, as Dechter and colleagues only support binary relations. If unordered, sequential activities are to be considered, approaches must be applied that enable the representation of non-binary, temporal relations, such as the frameworks described by Stergiou and Koubarakis, or Staab [25, 26]. However, the problem of verifying a guideline, that includes unordered sequential activities, is NP-hard as opposed to an effort of $O(n^3)$ otherwise. As Asbru is currently the only representation format that considers unordered sequential activities, we chose to integrate this type of control flow in the verification process only through approximate checks. Hereby, the efficiency of our approach is not compromised.

1.3 The plan-representation language Asbru

Within the *Asgaard* project, a set of tasks and computational models is investigated to support the execution of clinical guidelines [21]. As a foundation for the associated task-specific problem-solving methods, the time-oriented, intention-based language *Asbru* has been developed that uses plans as the basic structure for representing guidelines [12]. As expressed by the property *time-oriented*, Asbru plans are associated with a temporal dimension. Among the full set of Asbru language elements, containing knowledge classes such as *preferences*, *intentions*, *conditions* and *effects*, the following two components are of particular interest in our context:

Time annotations

The general purpose of a time annotation is to constrain the temporal occurrence of plan elements (including plans themselves): The plan element has to start in a certain time interval, finish in another time interval and its duration has to be within certain limits.

A time annotation is defined by seven parameters: *reference point* (Ref), *earliest starting shift* (ESS), *latest starting shift* (LSS), *earliest finishing shift* (EFS), *latest finishing shift* (LFS), *minimum duration* (minDu) and *maximum duration* (maxDu). These parameters are combined in the data structure $[[ESS, LSS], [EFS, LFS], [minDu, maxDu], Ref]$, where the starting- and finishing intervals represent offsets to the reference point. For their domains, the obvious bounds are used: The duration interval must be a subset of $[0, \infty)$, whereas the starting- and finishing intervals must be subsets of $(-\infty, \infty)$. A graphical representation of a time annotation on a one-dimensional scale is shown in Figure 1.

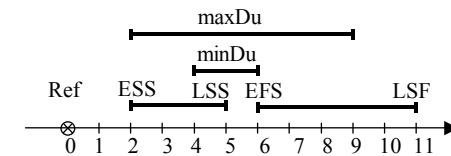


Figure 1. Graphical representation of time annotation $[[2; 5]; [6; 11]; [2; 7]; 0]$. Duration bars must be imagined as "floating" above the starting and finishing intervals, their position depicted here builds only one possible scenario.

By defining three time intervals from which the time annotation's starting and ending time points and duration may originate, Asbru allows a representation of temporal uncertainty. As each time annotation can further define its local reference point, multiple time lines are enabled. To support incomplete, temporal information, parameters may be left unrestricted, denoted by "_". Logically, this corresponds to setting the parameter to the respective bound of its interval's domain, i.e. an unrestricted minDu is set to 0, ESS to $-\infty$, and LFS to ∞ .

Note that none of the intervals is redundant. Any change of the intervals' parameters has an influence on the time annotation's characteristics as long as it stays within the bounds $[ESS^-, LSS^+]$, $[EFS^-, LFS^+]$ and $[minDu^-, maxDu^+]$ described in the following:

1. For each single time point in the starting interval $[ESS, LSS]$ there exists at least one duration out of $[minDu, maxDu]$, which allows the finishing interval $[EFS, LFS]$ to be reached, i.e. it is an element of $[ESS^-, LSS^+] = \{x \mid \exists y \in [minDu, maxDu], \exists z \in [EFS, LFS], x+y=z\}$.
2. Each single time point in the finishing interval $[EFS, LFS]$ is reachable by at least one duration out of $[minDu, maxDu]$ from the starting interval $[ESS, LSS]$, i.e. it is an element of $[EFS^-, LFS^+] = \{z \mid \exists x \in [ESS, LSS], \exists y \in [minDu, maxDu], x+y=z\}$.
3. Each duration connects one point from the starting interval with one point from the finishing interval, i.e. is element of $[minDu^-, maxDu^+] = \{y \mid \exists x \in [ESS, LSS], \exists z \in [EFS, LFS], x+y=z\}$.

As we will describe in section 1.4, a time annotation can be seen as a *temporal constraint network* with three variables for *Ref*, the *start* and the *finish* of an event, and the three intervals building distance constraints between them. In this interpretation, the above three restrictions on the intervals' bounds correspond to the properties of a *minimal network*, which provides the most explicit encoding of the involved constraints [2].

If at least one parameter of the starting or finishing intervals is defined together with a reference point, we say that a *time annotation*, respective *the plan element it refers to*, is *linked to the time line*. This means that it (respective the plan element) must not start or finish before or after a certain point in time. If only the duration interval is defined, however, the time annotation is completely detached from the time line. It (respective the plan element it refers to) may be executed anytime, as long as its duration stays within the specified bounds.

Plan Body

In Asbru, a clinical guideline is modeled as a set of hierarchical plans. As indicated by the term *hierarchical*, each plan may contain any number of subplans within its plan body, which may themselves be decomposed into sub-subplans, and so on. 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*, and use it as a synonym for a *clinical guideline*, coded in the *Asbru language*. Each plan corresponds to a single guideline activity (e.g., the activity *weaning* is represented as one of several plans within a guideline for the artificial ventilation of neonates, coded in Asbru). Note that the root plan of a plan hierarchy builds the top-level activity and is thus usually named after the guideline itself (e.g., plan *artificial ventilation of neonates*).

The control flow within a plan hierarchy is specified explicitly and allows sequential (with defined or undefined order), parallel, cyclical, and arbitrary execution of plans. During run-time, a plan is decomposed into its subplans until a non-decomposable plan - called *action* - is found. Actions are then transferred to suitable agents for their execution.

In order to constrain its possible execution intervals, a plan may be associated with a time annotation. The resulting data structure is called a *plan activation (PA)*. Within a *PA*, a plan may only be associated with one single time annotation, disjunctions of several time annotations are not allowed.

For example:

```
(GDM-II [[0 WEEKS, 8 WEEKS], [24 WEEKS, _], [18 WEEKS, _], CONCEPTION])
```

This Asbru code means: "Plan *GDM-II* should be executed in a time interval, *starting between 0 and 8 weeks after the conception, finishing no earlier than 24 weeks after the conception, and lasting at least 18 weeks.*" There are no restrictions on the latest finishing point and the maximum duration.

Figure 2 shows a plan hierarchy consisting of eleven plans in four layers, each of which is associated with a specific time annotation and such builds a *PA*. The execution of the plans within the hierarchy is further influenced by five different operators for flow control. The semantics of these operators and the additional parameters of the cyclical operator will be explained in section 2.1.2. The actual guideline actions (such as recommendations to the user) reside in the plan bodies of the leaf plans. As they are not essential for our task, they are omitted in Figure 2.

```
(P1 [[_, _], [_, 390], [50, 370], Ref]
  do-seq-ordered ((P2 [[_, _], [_, _], [70, 100], _]),
                 (P3 [[_, _], [_, _], [120, 150], _]),
                 (P4 [[_, _], [_, _], [130, 160], _])))

(P2 do-parallelly ((P5 [[_, _], [_, _], [90, _], _]),
                  (P6 [[40, _], [_, _], [_, _], Ref])))

(P3 do-cyclically ((P7 [[_, _], [_, _], [20, _], _]
                    retry=[10, _] exec=[5, _])))

(P4 do-arbitrarily ((P8 [[_, _], [_, _], [120, 160], _]),
                  (P9 [[_, _], [_, _], [20, _], _])))

(P8 do-seq-unordered ((P10 [[_, _], [_, _], [90, _], _]),
                     (P11 [[_, _], [_, _], [80, _], _])))
```

Figure 2. A plan hierarchy consisting of 11 *PAs* in 4 levels. Control flow is specified by 5 different operators.

1.4 Analogy to a simple temporal problem

Our verification task can be mapped to a simple temporal problem: According to Dechter and colleagues, a *simple temporal problem (STP)* is a temporal constraint satisfaction problem in which all involved (quantitative) constraints consist of a single interval [2]. This is a special case of the general *temporal constraint satisfaction problem*, where constraints are disjunctions of intervals.

The constraint network of an *STP* can be represented by a directed *constraint graph*: Nodes represent variables $X_1, \dots, X_n$ for the time point of the occurrence of certain events, whereas an edge $i \rightarrow j$ indicates that an interval $[a_{ij}, b_{ij}]$ for the temporal distance between the two corresponding events is specified representing the constraint $a_{ij} \leq X_j - X_i \leq b_{ij}$. Within the constraint network of an *STP*, no more than one interval per edge is allowed, which means that disjunctions of constraints are ruled out.

Each constraint can be expressed by a pair of inequalities $X_j - X_i \leq b_{ij}$ and $X_i - X_j \leq -a_{ij}$. This builds the basis of the directed *distance graph* (*DG*), which has the same node set as the *constraint graph*. Each edge, $i \rightarrow j$, however, is labeled by a weight a_{ij} representing the linear inequality $X_j - X_i \leq a_{ij}$.

A *PA* with a fully specified time annotation constrains the start, finish and duration of a plan. As it does not allow disjunctions of time annotations, it satisfies the corresponding demand of an *STP*. In order to represent a *PA* as the constraint network of an *STP*, we decompose it into the three nodes *Start of Plan* (*P.S*), *Finish of Plan* (*P.F*) and *Reference Point* (*Ref*). Figure 3 shows the constraint and distance graphs corresponding to a *PA*.

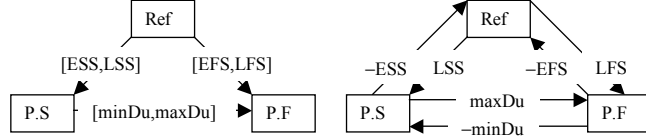


Figure 3. Constraint graph (left) and distance graph (right) of plan activation (P [ESS, LSS], [EFS, LFS], [$minDu, maxDu$], Ref)).

2 Verification of scheduling constraints

The *PA*s within an Asbru plan hierarchy build the target of our verification approach. They are checked for the following two properties:

- **Consistency**

We examine whether *PA*s are consistent with other scheduling constraints within a guideline, induced by control flow operators or hierarchical structuring. To cite an example, our goal is to check whether the *PA*s within a plan hierarchy, such as the one depicted in Figure 2, are consistent. The concept of consistency we use is based on the absence of negative cycles within the *DG* of an *STP*, as defined by Dechter and colleagues [2]: We call a *PA consistent with respect to its associated time annotation*, if and only if its *DG*-representation does not contain negative cycles. A negative cycle is defined as a cyclical path within the *DG*, for which the sum of weights is negative. Within the context of this paper, we will refer to a *PA, consistent with respect to its associated time annotation*, by the shorter term *consistent PA* only.

- **Minimality**

We examine whether each *PA* is *minimal with respect to its time annotation*. Using the abbreviated version *minimal PA* for the rest of the paper, we call a *PA minimal* if and only if the constraint network, given by the three intervals of the associated time annotation, is minimal. The concept of minimality of a constraint network is defined analogously to Dechter and colleagues [2]: A constraint network is minimal if it is *tighter* than any *equivalent* constraint network. Two networks are *equivalent* if they represent the same solution set, which in our case corresponds to two *PA*s that allow the same set of execution intervals for their plans. A network T is *tighter* than a network S if all constraints of T are tighter than the corresponding constraints in S , i.e. constraint T_{ij} is tighter than S_{ij} for all pairs i, j . A constraint T_{ij} , limiting the distance between two variables, is tighter than a constraint S_{ij} , if every pair of values allowed by T_{ij} is also allowed by S_{ij} . In other words, a *PA* is minimal, if its time annotation represents the most explicit encoding of the induced constraints.

Inconsistencies are treated as errors that have to be cleared by the user. Non-minimalities, however, represent sub-optimal specifications of *PA*s only. They are therefore seen as disfigurements of the plan hierarchy only, which are indicated to the user, but whose correction is not enforced.

As the presence of a negative cycle indicates that an *STP* is inconsistent, we will examine in section 2.1 how cycles may arise within one *PA* or within a network of several *PA*s. For each identified type of cycle we will examine how it may become negative and thereby introduce an inconsistency. In section 2.2, we will describe how single *PA*s and networks of several *PA*s may become non-minimal. Asbru's operator for unordered, sequential execution of plans does not fit into our framework, and will therefore be separately handled in section 2.3. There we will explain how our approach may partially cover this operator by using some approximate checks on its consistent application. In section 2.4, we will address the problem of multiple time lines within different *PA*s that complicates verification.

2.1 Origin of inconsistencies

As the presence of a negative cycle indicates that an *STP* is inconsistent, we will examine in the sequel how cycles may arise within one *PA* or within a network of several *PA*s. For each identified type of cycle we will examine how it may become negative and thereby introduce an inconsistency.

In the general case an inconsistency will originate from a cycle, involving several *PA*s. The linking of these *PA*s is caused by the semantics of certain Asbru language elements and will be explained in sections 2.1.2, 2.1.3, and 2.1.4.

2.1.1 Single plan activation

If we take a look at the *DG* of a single, fully specified *PA* (see Figure 3), we can identify cycles already here.

First we have three cycles, each enclosing two of the three nodes: ESS and LSS build a cycle between Ref and $P.S$, EFS and LFS between Ref and $P.F$, and finally $minDu$ and $maxDu$ build a cycle between $P.S$ and $P.F$. A trivial kind of inconsistency will occur here, if at least one of the following inequalities holds:

$$ESS > LSS, EFS > LFS \text{ or } minDu > maxDu$$

Second we receive two cycles comprising ESS , LFS and $minDu$, respective LSS , $maxDu$ and EFS that span all three nodes of a *PA*. All other cycles that span all three nodes are concatenations of two two-node cycles and may be correspondingly decomposed. Inconsistencies resulting from the latter two cycles will occur, if at least one of the following inequalities holds:

$$minDu > LFS - ESS \text{ or } EFS - LSS > maxDu$$

Looking for relative positionings of the starting and finishing intervals that can be excluded a priori as being inconsistent, the latter two inequalities give us the following trivial insight: Considering that the minimum value for $minDu$ is 0, the first inequality tells us that a finishing interval, that is in relation *before* to the starting interval, is always inconsistent. The second leaves any position of the starting interval relatively to the finishing interval open, as $maxDu$ is unbounded on the upper end.

2.1.2 Specification of control flow

As mentioned in section 1.3, the Asbru languages provides operators that allow the user to explicitly define the order in which plans are to be executed. We will now examine what kind of constraints between *PAs* are imposed by the semantics of these operators for flow control.

- Sequential execution of plans with defined order

Asbru provides an operator for sequential, ordered execution of plans with the common semantics: Every plan, activated as the $(i+1)^{\text{th}}$ member of a set of sequential plans with defined order, must not be started before the plan, corresponding to the i^{th} member, is finished. In other words, a sequential, ordered execution of plans P1 and P2 implies the qualitative constraint "P1.F $\leq$ P2.S" between the finish of P1 and the start of P2. This translates to the quantitative constraint $[0, \infty)$ between time points P1.F and P2.S [11].



Figure 4. Constraint implied by sequential, ordered execution of plans P1 and P2.

Figure 4 visualizes this circumstance in the constraint and distance graphs. The maximum distance of ∞ is omitted in the *DG* as it does not represent a constraint.

- Parallel execution of plans

The Asbru operator for parallel plan execution has the following semantics: Parallel plans have to start concurrently, whereas no restriction is made for their finishing time point. This entails the qualitative constraint "P1.S = P2.S" between the starting time points of parallel plans P1 and P2, which translates to a maximal (and therefore also a minimal) distance of 0 between these points.

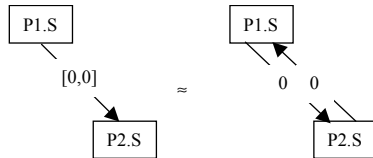


Figure 5. Constraint implied by parallel execution of plans P1 and P2.

Figure 5 shows the graphical representation of this constraint for two parallel plans. The arc is presented in diagonal orientation to optically distinguish it from *parent – child* relationships.

- Arbitrary execution of plans

The operator for arbitrary execution of plans does not impose any constraint on the order: Plans may be executed in parallel, overlapping or sequentially. Therefore, the distance between them cannot be restricted in any way.

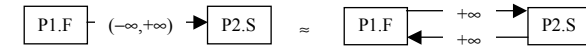


Figure 6. Constraint implied by arbitrary execution of plans P1 and P2.

For reasons of completeness, Figure 6 visualizes the unconstrained distance between the start and the finish of two arbitrary executed plans.

- Cyclical execution of plans

The Asbru language offers several ways to specify a repetitive execution of a single plan. All these alternatives are extensions of the following scheme: A regular *PA* is enhanced by a retry-delay and a lower (*minExec*) as well as an upper limit (*maxExec*) for the number of plan executions. The retry-delay is given by an interval that defines the minimum (*minDelay*) and maximum waiting period (*maxDelay*) between two plan executions.

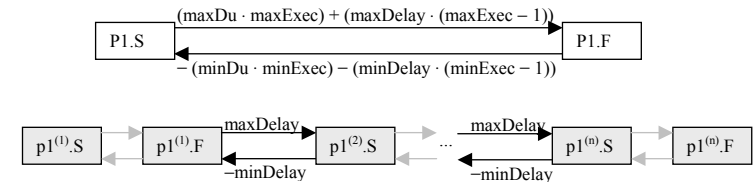


Figure 7. Static view of cyclical plan P1 with instance p1 below. Instance p1 is executed n times, where $minExec \leq n \leq maxExec$

Figure 7 shows the *DG* of the cyclical execution of a single plan. The upper graph gives a static view of the cyclical plan, whereas the lower graph shows one particular instance in detail. Minimum and maximum durations are represented in gray. As the retry-delay and number of plan executions can be defined by the user, they are susceptible to inconsistencies. We mentioned in section 2.1 that a *PA* is inconsistent if $minDu > maxDu$. Analogous, a cyclical *PA* is inconsistent if $(minDu \cdot minExec) + (minDelay \cdot (minExec - 1)) > (maxDu \cdot maxExec) + (maxDelay \cdot (maxExec - 1))$.

2.1.3 Parent – child relations

We already stated in section 1.3 that Asbru organizes guidelines in a tree structure of plans called *plan hierarchy*. A plan hierarchy may consist of several levels, and each non-leaf plan has a *parent* → *child* relation to the plans of the next lower level. Using an obvious approach of parent – child synchronization, Asbru does not allow child plans to start before their parent or finish after their parent. For a parent plan P and a child plan C this entails the qualitative constraints "P.S ≤ C.S" and "C.F ≤ P.F" between their starting and finishing time points. This translates to a minimal distance of 0 between P.S and C.S and a minimal distance of 0 between the C.F and P.F.

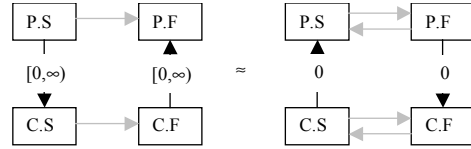


Figure 8. Constraints between parent plan P and child plan C.

Figure 8 visualizes this circumstance in the constraint and distance graphs, where duration constraints respective distances are shown in gray to make cycles obvious. Clearly, the *PAs* of plans P and C will be inconsistent, if $C.minDu > P.maxDu$.

If we combine the constraints, originating from control flow with those originating from parent – child relationships, we receive less obvious possibilities for inconsistencies: Figure 9 depicts a scenario, where plan P1 has two child plans P2 and P3, which are executed *sequentially* in order P2, P3. Plan P2 itself is parent of the *cyclical* plan P4.

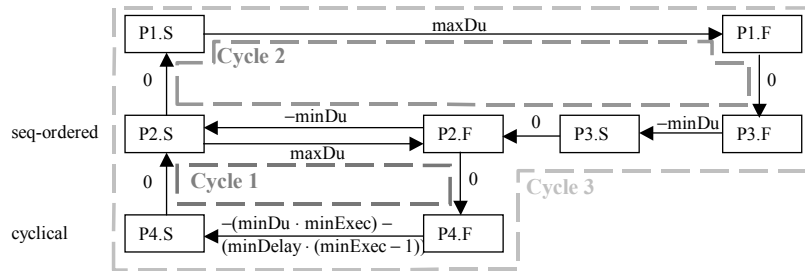


Figure 9. Cycles involving parent-child relations and the control flow.

The scenario in Figure 9 contains three different cycles. Cycle 1 between P2 and P4, shown in dark, may induce an inconsistency between the *PAs* of P2 and P4, if $P4[(minDu \cdot minExec) + (minDelay \cdot (minExec - 1))] > P2.maxDu$. Cycle 2 between P1, P2 and P3, represented in medium gray, may yield an inconsistency if $P2.minDu + P3.minDu > P1.maxDu$. Cycle 3 is depicted in light gray and circumvents the minimum duration of P2 by passing over P4. It will be inconsistent, if $P3.minDu + P4[(minDu \cdot minExec) + (minDelay \cdot (minExec - 1))] > P1.maxDu$. Note that the third cycle is not made obsolete by the other two: The total minimum duration of all instantiations of P4 can be lower than the maximum duration of P2 (hereby avoiding an inconsistency in cycle 1) but higher than the minimum duration of P2 at the same time. Together with the minimum duration of P3, it may then exceed the maximum duration of P1, causing an inconsistency in cycle three.

2.1.4 Links to the time line

We have seen in sections 1.4 and 2.1.1 how links to the time line can create cycles within a single *PA*. Now we will demonstrate that such cycles may also span several *PAs*. Depending on the involved control flow operators, different kinds of cycles are possible. Therefore, we will examine in the following for each control flow operator, how links to the time line within the involved *PAs* may induce inconsistencies:

• Sequential execution of plans with defined order

Assume a set of sequential plans with defined order, where an early executed plan P_m has a link to the time line of type "earliest" (ESS or EFS defined) and a later executed plan P_n has one of type "latest" (LSS or LFS defined) with $m < n$. Then these links form a cycle within the corresponding distance graph, which further involves the minimum durations of the intermediate plans, the minimum durations of 0 between the finish and the start of neighboring plans and potentially the minimum durations of P_m and P_n .

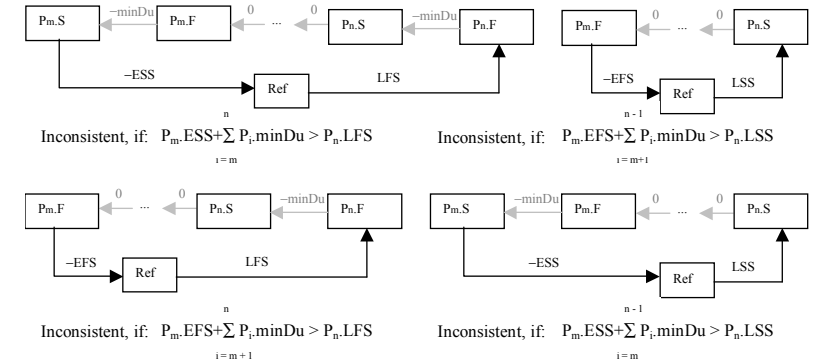


Figure 10. Inconsistent cycles introduced by sequential plans linked to the time line.

We omit cycles in the opposite direction as they involve the unconstrained maximum distances of ∞ between the finish and the start of consecutive plans. Figure 10 shows the possible scenarios with grayed minimum duration constraints and the conditions for each cycle to become inconsistent.

- Parallel execution of plans

Assume a set of parallel plans including plans P_i and P_j , which have links to the time line of opposite type, i.e. one of type "earliest", the other of type "latest". Then we distinguish four types of cycles induced by these links. Figure 11 shows the scenarios with grayed duration constraints and the conditions for each cycle to become inconsistent.

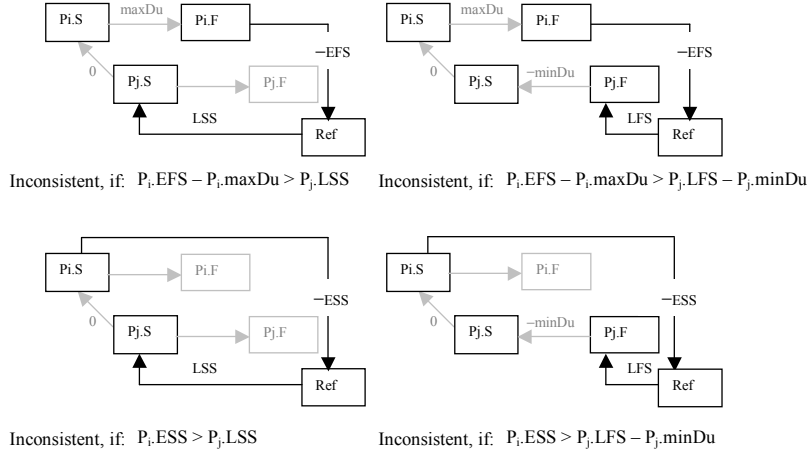


Figure 11. Inconsistent cycles introduced by parallel plans linked to the time line.

- Arbitrary execution of plans

As the minimum and maximum distances in the DG between two arbitrarily executed PAs is $+\infty$, a cycle involving two or more such plans (of the same parent) cannot become negative. Inconsistencies can only originate from cycles that involve the links between a single arbitrary plan and its parent or children plans.

- Cyclical execution of plans

If a cyclical PA contains a starting and/or a finishing interval, these refer to the start of the first execution respective to the finish of the last execution of the plan. As already shown, the total minimum and maximum durations between the start of the first and the finish of the last executions are $(minDu \cdot minExec) + (minDelay \cdot (minExec - 1))$ and $(maxDu \cdot maxExec) + (maxDelay \cdot (maxExec - 1))$. As we thus treat all executions of a cyclical PA as a whole and its links to the time line can only refer to the first and last executions, only a single PA is involved here. Therefore, a cyclical PA linked to the time line can only be part of a network of PAs through the links to its parent or children plans, it cannot generate a network of PAs itself. When examining a network of PAs for inconsistencies, cyclical plans therefore only have to be considered as part of those cycles, which include the cyclical plan's links to its parent and/or children plans. However, each single cyclical plan can become inconsistent itself as shown in section 2.1.2.

2.2 Origin of non-minimalities

In this section, we will examine how non-minimalities may arise within one PA or within a network of several PAs . Although we do not allow inconsistent PAs within a plan hierarchy, we do not force the user to define minimal PAs during the design of a plan hierarchy. Ensuring manually that each PA is minimal would probably be too annoying for the user. Instead, we point out non-minimal PAs to the user during plan verification, and give her the option to adapt it. Providing the minimal variant of a PA as a suggestion on how to modify it does not require an extra effort: The algorithm we use for consistency checking is based on the computation of minimal PAs within a plan hierarchy.

2.2.1 Single plan activation

A single PA , which is consistent by not containing a negative cycle described in section 2.1.1, is not necessarily minimal. It will be minimal, if it further satisfies the following additional demands:

- $ESS \leq EFS$ and $LSS \leq LFS$
- $MAXIMUM(0, EFS - LSS) \leq minDu \leq MINIMUM(EFS - ESS, LFS - LSS)$
- $MAXIMUM(EFS - ESS, LFS - LSS) \leq maxDu \leq LFS - ESS$

The first point allows the starting interval to be in any of the relations *{equal, overlaps, meets, before}* to the finishing interval. Depending on the starting and finishing intervals, the second and third points yield the bounds for the duration interval. On the left side of Figure 12, a PA is shown that does not satisfy the above demands. The PA on the right side of Figure 12 represents an equivalent, minimal version of the same PA that satisfies the above demands. We use the representation introduced in Figure 1 here, as it allows a graphical visualization of the difference between non-minimal and minimal PAs .

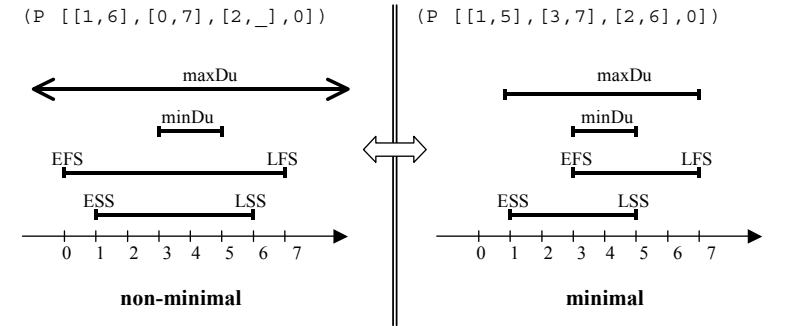


Figure 12. Two equivalent PAs , left the non-minimal version, right the minimal version.

2.2.2 Network of plan activations

Non-minimality of a PA may also be induced by links to other PA s in a common network. The left side of Figure 13 shows a plan hierarchy containing PA s, which are rendered non-minimal by two types of scheduling constraints: One is implied by a control flow operator for sequential execution and the other by a parent – child relation.

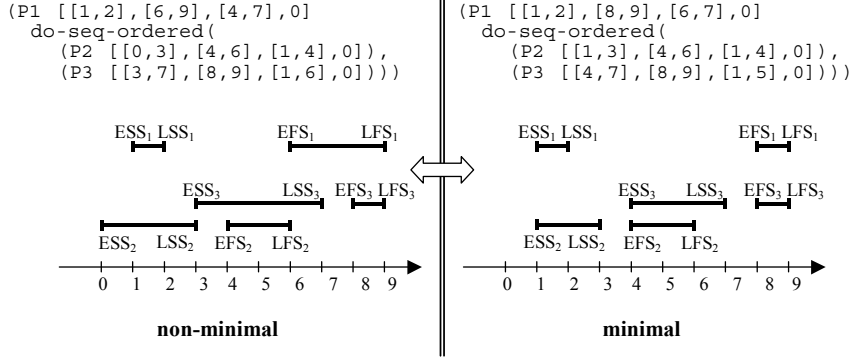


Figure 13. Two equivalent networks of PA s, left the non-minimal version, right the minimal version. Duration intervals are omitted.

2.3 Handling Asbru's operator for unordered, sequential plan execution

The Asbru language comprises an operator for a sequential execution of plans, where the order in which the plans are executed is not specified. Accordingly, an unordered, sequential execution of plans P1 and P2 presumes that P1 executes either before or after P2. In terms of the DG this means that there is a minimal distance of 0 between either the finish of P1 and the start of P2, or between the finish of P2 and the start of P1.

The disjunctions within the last two sentences already indicate that an unordered, sequential execution of plans goes beyond the scope of $STPs$. Unfortunately it also does not fit into the framework of general temporal constraint satisfaction problems ($TCSP$ s) as defined by Dechter and colleagues [2]: Although their $TCSP$ framework allows disjunctions within constraints, the latter may only be of binary type. This means that each disjunct within one disjunction must exclusively refer to the same pair of variables (e.g., $a \leq X_j - X_i \leq b \vee c \leq X_j - X_i \leq d$). However, the unordered, sequential execution of plans P1 and P2 introduces the 4-ary constraint $0 \leq P2.S - P1.F \leq \infty \vee 0 \leq P1.S - P2.F \leq \infty$.

To incorporate unordered, sequential execution of plans, we need an approach that can handle non-binary constraints, such as the framework proposed by Stergiou and Koubarakis [26]. They consider temporal constraints of the form $X_i - Y_j \leq r_1 \vee \dots \vee X_n - Y_n \leq r_n$, where $X_1, \dots, X_n, Y_1, \dots, Y_n$ are variables and $r_1, \dots, r_n$ are constants. This covers the representation of unordered, sequential plans, such as for plans P1, P2 and P3 we would define the three constraints $(P1.F - P2.S \leq 0) \vee (P2.F - P1.S \leq 0)$, $(P1.F - P3.S \leq 0) \vee (P3.F - P1.S \leq 0)$ and $(P2.F - P3.S \leq 0) \vee (P3.F - P2.S \leq 0)$.

However, the problem of determining the consistency of a set of constraints, that includes disjunctions of non-binary constraints, is NP-hard. Therefore, we choose to reach an approximate solution for unordered, sequential plans only, by checking whether there are any easy to find objections on their consistency:

- We check whether there is any objection to the assumption that the set of plans can be executed in sequential order at all. This is the case if two or more of the corresponding PA s are *overlapping*, which means that they force the plans to be executed at least partially in parallel. Whether two PA s are overlapping can be deduced from their parameters LSS and EFS:

$$\text{Overlapping}(PA_1, PA_2) \text{ iff } ((LSS(PA_1) < EFS(PA_2)) \wedge (LSS(PA_2) < EFS(PA_1)))$$

If two or more PA s of a set of unordered, sequential plans are overlapping, we will receive an inconsistency (i.e. a negative cycle within their distance graph) no matter how we order them.

- The constraint on the execution interval of a child plan with respect to its parent's execution interval (see section 2.1.3) clearly also holds for unordered, sequential plans. The corresponding constraints are inserted in the distance matrix accordingly.
- The total minimum durations of a set of unordered, sequential plans must not exceed their parent's maximum duration. To perform the corresponding check, we can execute an extra verification run, where we treat the set of unordered, sequential plans as ordered, sequential plans, considering only their duration intervals but ignoring their starting and finishing intervals.

2.4 Multiple time lines complicate verification

As we have seen in section 2.1.4, our verification process involves checking whether particular constraints between two or more *PAs*, which are linked to the time line, are consistent. This aim is complicated by Asbru's ability to support multiple time lines within one plan hierarchy:

Asbru allows the reference point of a *PA* to refer to patient-specific time points (e.g., BIRTH, CONCEPTION, ...). The actual calendar date for such a time point can only be determined during the execution of a plan, after it has been assigned to a patient. Statically, its relation to another *PA*'s reference point is a priori unknown, which means that the distance between them is set to $(-\infty, +\infty)$. This, in due course, can prevent certain "obvious" inconsistencies from being detected:

Assume, we have to check in the context of a guideline that treats gestational diabetes, whether plans P1 and P2 can be executed sequentially in order P1, P2, when activated by (P1 $[[_], _], [2, _], [_, _], \text{delivery}]$) and (P2 $[[_], 2], [_, _], [_, _], \text{conception}]$) (see upper diagram in Figure 14). As the unknown distance between the two reference points turns the sum of weights of the induced cycle to $+\infty$, a sequential execution of P1 and P2 is found to be consistent (see lower left diagram in Figure 14).

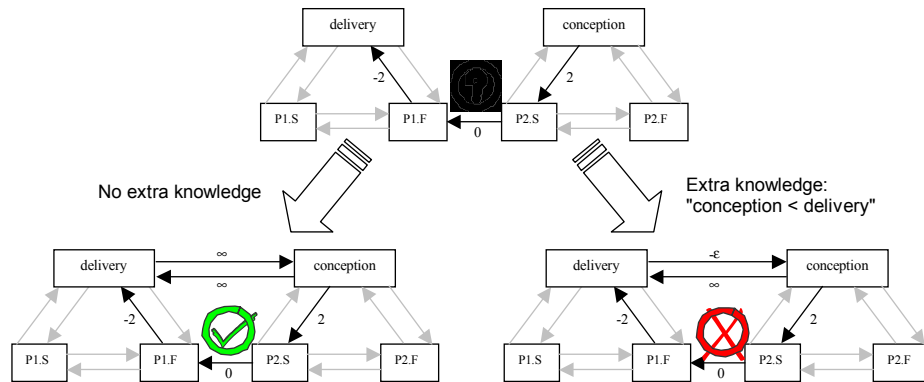


Figure 14. Within a guideline treating gestational diabetes, it is checked whether plans P1 and P2 may be executed sequentially in order P1,P2, when activated by (P1 $[[_], _], [2, _], [_, _], \text{delivery}]$) and (P2 $[[_], 2], [_, _], [_, _], \text{conception}]$). Without extra knowledge, this scenario is found to be consistent. When the extra knowledge "conception < delivery" is included, the scenario is found to be inconsistent.

In such a case, verification can be optimized if extra knowledge on the relation between the reference points of different *PAs* is available. Assume, we include the knowledge in our checking process, that for the same pregnancy the event *delivery* always happens *after* the event *conception*. This translates to a distance of $[\epsilon, +\infty)$, where ϵ represents the smallest, supported time unit. When using this extra knowledge, a sequential execution of P1 and P2 is found to be inconsistent (see lower right diagram in Figure 14).

No extra knowledge is needed when checking *PAs* with identical reference points, as they are trivially related. The same holds for *PAs*, whose reference points are specified as time points of the calendar.

3 Method

Analogous to Dechter and colleagues, we use Floyd-Warshall's *all-pairs-shortest-paths algorithm* to detect inconsistencies within a network of *PAs* (see Figure 15). Applied to the network's *DG*, the algorithm calculates the minimal network represented as a matrix of minimal distances d_{ij} in time $O(n^3)$, where n is the number of nodes. Inconsistencies (i.e. negative cycles) can simply be detected by examining the sign of the diagonal elements d_{ii} within the minimal distance matrix.

```

for i = 1 to n do  $d_{ii} \leftarrow 0$ ;
for i,j = 1 to n do  $d_{ij} \leftarrow a_{ij}$ ;
for k = 1 to n do
  for i,j = 1 to n do
     $d_{ij} \leftarrow \min(d_{ij}, d_{ik} + d_{kj})$ ;

```

Figure 15. Floyd-Warshall's algorithm.

The matrix of distances a_{ij} is set up by inserting the defined constraints originating from the individual *PAs*, parent-child relations and control flow operators. Instead of the value ∞ for unconstrained distances, we use the constant $(n \cdot \text{MAX}(a_{ij})) + 1$, as no path can have more than n arcs.

The algorithm further enables the identification of non-minimal *PAs* and provides their minimal versions as a suggestion on how to adapt them. We make use of this feature by pointing out non-minimal *PAs* to the user during the verification process and leave her the choice to accept the minimal version, modify the *PA* differently or leave it as it is. An extra benefit of calculating the minimal network is the fact that it may be used to assemble solutions, i.e. assignments to the parameters of all *PAs*, that are consistent with the network's constraints in time $O(n^2)$. Finding a solution of the network is an essential task for the guideline interpreter to determine the execution intervals of all plans.

4 Results

Up to now, four existing clinical guidelines have been represented in the Asbru language. These are a guideline for the observation and treatment of gestational diabetes mellitus used at the Stanford Medical School, a guideline that supports the artificial ventilation of neonates used at the neonatal intensive care unit of the University of Vienna Medical School, a guideline for the treatment of sinusitis used by general practitioners in the Netherlands, and a guideline for the management of hyperbilirubinemia in newborns issued by the American Academy of Pediatrics. Unfortunately, none of them utilizes *PAs* to an extent that would require an automatic verification of their time annotations. Therefore, we will demonstrate the application of our approach by verifying the *PAs* within the fictive guideline shown in Figure 2.

To visualize the different cycles, emerging from this guideline, we depict the corresponding *DG* in Figure 16. It does not include the two unordered sequential plans P10 and P11, as the corresponding constraints on their control flow cannot be represented within a *DG*.

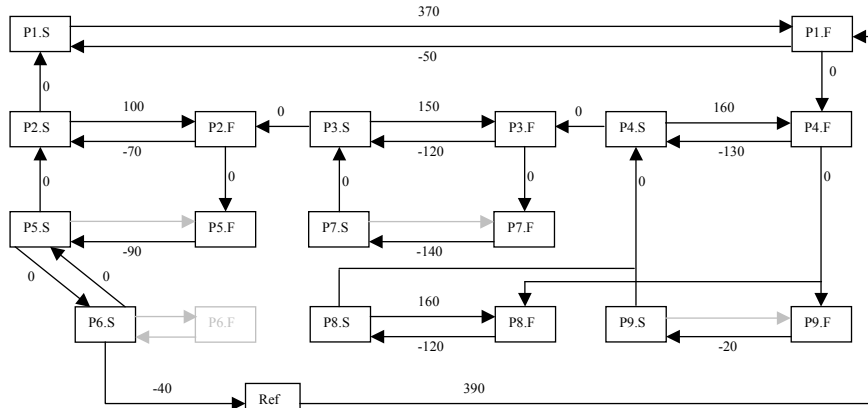


Figure 16. *DG* of the plan hierarchy, shown in Figure 2.

We implemented a prototype verifier that is based on the method described in section 3. The application contains two text areas, where the left shows the plan hierarchy to be analyzed, and the right depicts the results of the verification process.

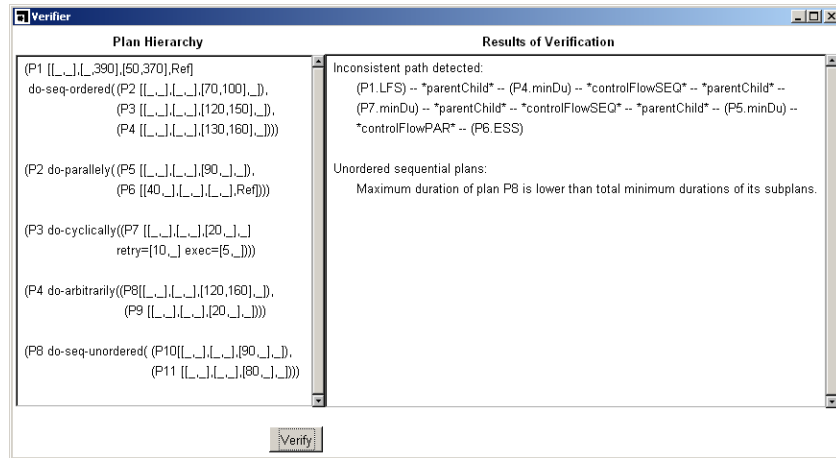


Figure 17. Result of verifying the plan hierarchy, shown in Figure 2.

Figure 17 shows the result of verifying the fictive guideline of Figure 2.

It tells us that the *PAs* within the plan hierarchy are inconsistent because of two reasons: First, an inconsistent path, corresponding to a negative cycle within Figure 16, is present, which involves the minimum durations of plans P4, P7 and P5, the latest finishing shift of P1, the earliest starting shift of P6, and several links induced by control flow operators and parent-child relations. Second, the total minimum duration of the unordered sequential plans P10 and P11 exceeds the maximum duration of their parent plan P8.

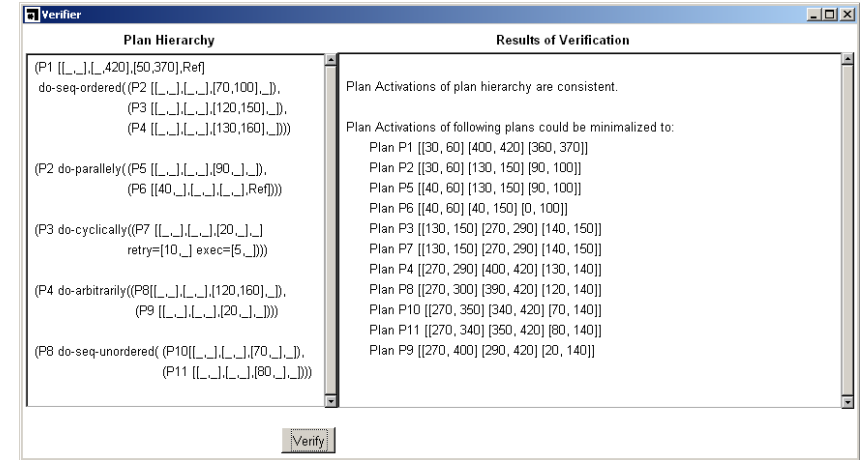


Figure 18. Result of verifying the corrected plan hierarchy.

Figure 18 shows the result of verifying the guideline after the two inconsistencies were cleared by setting the latest finishing shift of plan P1 to 420 and the minimum duration of plan P10 to 70. As described in section 2.2, the verifier provides the minimal version of each *PA* as a suggestion on how to adapt it.

5 Conclusion

Scheduling constraints, expressed either explicitly by control flow operators (e.g., for sequential or parallel execution) or implicitly by a hierarchical modeling, are basic elements of most current formats for the computerized representation of clinical guidelines. Another type of scheduling constraint, which can be frequently found in conventional guidelines, is given by temporal specifications (e.g., "determine the newborn's blood glucose level within 30 minutes after birth", "execute a controlled ventilation for at least 3 hours"). The importance of the latter kind of constraint is reflected by a growing number of integrations within current guideline specification formats. However, like other types of knowledge, temporal scheduling constraints may be used inconsistently within a guideline and may thus represent an obstacle for the guideline's computerization.

In this paper we have addressed this problem by presenting an approach for the verification of temporal scheduling constraints within clinical practice guidelines, represented within the Asbru language. The approach is based on the calculation of the minimal, temporal network, and serves three purposes: (1) It allows the identification of inconsistent temporal scheduling constraints, (2) It detects non-minimal temporal scheduling constraints and yields suggestions for their representation in an equivalent, but more explicit form, (3) It can be used by the guideline interpreter to determine feasible execution intervals for each guideline activity.

The computation of the minimal network can be done in $O(n^3)$ time for a guideline with n individual activities and therefore constitutes an efficient method for its verification. Efficiency is an issue for us insofar as it allows our verification process to be applied interactively during the design phase of a guideline, e.g. after each definition of a new temporal scheduling constraint, without annoying the user by long answering times.

A limitation of our approach consists in its inability to fully verify temporal constraints on the execution of unordered sequential guideline activities. Although we apply several checks that can detect basic types of inconsistencies in this case, a full verification would require a technique that can handle disjunctions of non-binary constraints. Even though suitable approaches for this task have been described in literature, e.g. [25, 26], the problem of fully verifying a guideline, that includes unordered sequential activities, is NP-hard. As Asbru is currently the only representation format that considers unordered sequential activities, we chose to integrate them in the verification process only through approximate checks. Hereby, the efficiency of our approach is not compromised.

Our approach is reusable in two ways: (1) It is applicable for the verification of several guideline-representation formats, other than Asbru. This is due to the fact that our approach assumes the existence of certain, basic operators for flow control within a guideline representation format (such as sequential, parallel or cyclical execution) and its organization in a hierarchical structure. These concepts are realized in most current approaches, e.g. [4, 7, 15, 16]. Our considerations on temporal scheduling constraints (originating from Asbru's PAs in this paper) have to be adapted when reusing our approach for a different representation, according to the expressiveness of the respective format in this area. (2) Our approach can be ported to other domains than guideline-based care, due to the applicability of Asbru in most areas of planning.

Acknowledgments. The authors thank Michael Balser, Frank van Harmelen, Peter Johnson, Robert Kosara, Wolfgang Reif, Andreas Seyfang, Yuval Shahar, and Annette ten Teije, who contribute to the development of the project. This project is supported by "Fonds zur Förderung der wissenschaftlichen Forschung" (Austrian Science Fund), P12797-INF.

6 References

- [1] J.F. Allen, Maintaining knowledge about temporal intervals, *Communications of the ACM* 26 (1983) 832-43.
- [2] R. Dechter, I. Meiri and J. Pearl, Temporal constraint networks, *Artificial Intelligence* 49 (1991) 61-95.
- [3] G. Duftschmid and S. Miksch, Knowledge-based verification of clinical guidelines by detection of anomalies, *Artificial Intelligence in Medicine* 22 (2001) 23-41.
- [4] J. Fox, N. Johns and A. Rahmzadeh, Disseminating medical knowledge: the PROforma approach, *Artificial Intelligence in Medicine* 14 (1998) 157-181.
- [5] C. Gordon, I. Herbert and P. Johnson, Knowledge representation and clinical practice guidelines: the DILEMMA and PRESTIGE projects, in: Brender-J, Christensen-J, Scherrer-JR and McNair-P, eds., *Proceedings of Medical Informatics Europe '96*; Copenhagen, Denmark (R. Brompton Hospital London UK. Medical Informatics Europe '96: Human Facets in Information Technologies. IOS Press Amsterdam Netherlands, 1996).
- [6] A. Guarnero, M. Marzuoli, G. Molino, P. Terenziani, M. Torchio and K. Vanni, Contextual and temporal clinical guidelines, in: *Proceedings of the AMIA-98 Annual Symposium* (1998) 683-687.
- [7] S. Herbert, C. Gordon, A. Jackson-Smale and S. Renaud, Protocols for clinical care, *Computer Methods and Programs in Biomedicine* 48 (1995) 21-26.
- [8] G. Hripcsak, Writing Arden Syntax medical logic modules, *Computers in Biology and Medicine* 24 (1994) 331-363.
- [9] H.A. Kautz and P.B. Ladkin, Integrating metric and qualitative temporal reasoning, in: *Proceedings 9th National Conference on Artificial Intelligence (AAAI-91)* (AT&T Bell Labs. Murray Hill NJ USA. AAAI Press Menlo Park CA USA, 1991).
- [10] C. McDonald and J. Overhage, Guidelines you can follow and trust: An ideal and an example, *Journal of the American Medical Association (JAMA)* 271 (1994) 872-873.
- [11] I. Meiri, Combining qualitative and quantitative constraints in temporal reasoning, *Artificial Intelligence* 87 (1996) 343-385.
- [12] S. Miksch, Y. Shahar and P. Johnson, Asbru: A task-specific, intention-based, and time-oriented language for representing skeletal plans, in: *Proceedings of the 7th Workshop on Knowledge Engineering: Methods & Languages (KEML-97)*, Milton Keynes, UK (1997).
- [13] D.W. Miller, Jr, S.J. Frawley and P.L. Miller, Using semantic constraints to help verify the completeness of a computer-based clinical guideline for childhood immunization, *Computer Methods and Programs in Biomedicine* 58 (1999) 267-280.
- [14] M. Musen, J. Rohn, L. Fagan and E. Shortliffe, Knowledge engineering for a clinical trial advice system: uncovering errors in protocol specification, *Bulletin du Cancer* 74 (1987) 291-296.
- [15] M. Musen, S. Tu, A. Das and Y. Shahar, EON: A component-based approach to automation of protocol-directed therapy, *Journal of the American Medical Informatics Association (JAMIA)* (1996) 367-388.
- [16] L. Ohno-Machado, J. Gennari, S. Murphy, N. Jain, S. Tu, D. Oliver, E. Pattison-Gordon, R. Greenes, E. Shortliffe and G. Barnett, The GuideLine Interchange Format: A model for representing

guidelines, *Journal of the American Medical Association (JAMA)* 5 (1998) 357-372.

[17] E. Pattison-Gordon, J. Cimino, G. Hripcsak, S. Tu, J. Gennari, N. Jain and R. Greenes, Requirements of a sharable guideline representation for computer applications. Stanford Technical Report SMI-96-0628, Stanford University, 1996.

[18] M. Peleg, A.A. Boxwala, O. Ogunyemi, Q. Zeng, S. Tu, R. Lacson, E. Bernstam, N. Ash, P. Mork, L. Ohno-Machado, E.H. Shortliffe and R.A. Greenes, GLIF3: the evolution of a guideline representation format, in: *Proceedings of the AMIA-2000 Annual Symposium*; Los Angeles, CA (2000) 645-649.

[19] S. Quaglini, R. Saracco, M. Stefanelli and C. Fassino, Supporting tools for guideline development and dissemination, in: *Proceedings of the 6th Conference on Artificial Intelligence in Medicine Europe (AIME)* (1997) 39-50.

[20] J.F. Rit, Propagating temporal constraints for scheduling, in: *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-86)*; Los Altos, CA (Morgan Kaufmann, 1986) 383-388.

[21] 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* 14 (1998) 29-51.

[22] E. Sherman, G. Hripcsak, J. Starren, R. Jender and P. Clayton, 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)* (1995) 238-242.

[23] R. Shiffman, Representation of clinical practice guidelines in conventional and augmented decision tables, *Journal of the American Medical Informatics Association (JAMIA)* 4 (1997) 382-393.

[24] R. Shiffman and R. Greenes, Improving clinical guidelines with logic and decision-table techniques, *Medical Decision Making* 14 (1994) 245-254.

[25] S. Staab, On non-binary temporal relations, in: Prade-H, ed. *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI 98)* (Comput. Linguistics Lab. Freiburg Univ. Germany. Wiley Chichester UK, 1998).

[26] K. Stergiou and M. Koubarakis, Backtracking algorithms for disjunctions of temporal constraints, *Artificial Intelligence* 120 (2000) 81-117.

[27] J. Stillman, R. Arthur and A. Deitsch, Tachyon: A constraint-based temporal model and its implementation, *SIGART Bulletin* 4 (1993).

[28] W. Tierney, J. Overhage, B. Takesue, L. Harris, M. Murray, D. Vargo and C. McDonald, Computerizing guidelines to improve care and patient outcomes: the example of heart failure, *Journal of the American Medical Informatics Association (JAMIA)* 2 (1995) 316-322.

[29] S. Vere, Planning in time: Windows and durations for activities and goals, *IEEE Transactions on PAMI* 5 (1983) 246-267.

[30] M. Vilain and H. Kautz, Constraint propagation algorithms for temporal reasoning, in: *Proceedings AAAI-86: Fifth National Conference on Artificial Intelligence*; Univ (BBN Labs. Cambridge MA USA. American Assoc. Artificial Intelligence Menlo Park CA USA, 1986).