A Verified, Executable Formalism for Resilient and Pervasive Guideline-Based Decision Support for Patients

Nick L. S. Fung[1](✉), Marten J. van Sinderen[1], Valerie M. Jones[1], and Hermie J. Hermens[1,2]

[1] University of Twente, 7500 AE Enschede, The Netherlands
{l.s.n.fung,m.j.vansinderen,v.m.jones}@utwente.nl
[2] Roessingh Research and Development, 7500 AH Enschede, The Netherlands
h.hermens@rrd.nl

Abstract. We present an executable formalism for clinical practice guidelines, with the aim of providing pervasive and evidence-based decision support to patients. Unlike traditional formalisms that capture the control flow between tasks, we focus on data flow, with tasks modeled as processes that execute in parallel. By parallelizing and distributing guideline knowledge, each device that constitutes the patient's pervasive healthcare system can provide decision support independently, avoiding single points of failure. This distribution also enables dynamic system re-configurations, increasing its resilience against evolving requirements and changing communications environments.

Our model recognizes four types of processes: Monitoring, Analysis, Decision and Effectuation. These processes were specified using (axiomatic) set theory and implemented as a set of libraries on top of Rosette, which supports execution of the formalism and verification of it using constraint solvers. The formalism was also tested by formalizing a complete clinical guideline for diabetes management, which yielded a Rosette program that was then tested on simulated patient data. The major point of clinical relevance is enhancing the quality and safety of decision support delivered to patients.

Keywords: Computerized clinical practice guidelines · Pervasive healthcare · Knowledge representation · Data flow modeling · Formal specification · Verification and validation · Diabetes management

1 Introduction

Pervasive healthcare systems, which aim to support patients anytime, anywhere, have potential to address the healthcare challenges arising from increased prevalence of chronic diseases, an aging population and shortage of healthcare resources [1]. For example, instead of relying on infrequent checkups, diabetic patients may use such systems to help constantly monitor and control their blood

M. Michalowski and R. Moskovitch (Eds.): AIME 2020, LNAI 12299, pp. 427–439, 2020.
https://doi.org/10.1007/978-3-030-59137-3_38

glucose levels. Such systems should, however, ensure high quality care, which in hospital settings is increasingly supported by the use of clinical practice guidelines (CPGs), especially computerized CPGs that can be executed automatically by knowledge-based systems (KBSs).

We aim to bring computerized CPGs to free-living settings to provide pervasive guideline-based decision support to patients. Clinical KBSs are typically deployed on fixed infrastructures, e.g. hospital servers, but for pervasive healthcare systems, components may be distributed across multiple personal devices and may require dynamic reconfiguration in response to changing requirements and unreliable communications environments. Here we present a new verified formalism for representing clinical guidelines which models clinical tasks as processes executing in parallel. In this way, guideline knowledge and reasoning can be flexibly distributed across system components, allowing them to operate independently and thereby avoid a single point of failure.

Section 2 presents background and related work while Sect. 3 presents our formalism. We present a reference implementation of the formalism in Sect. 4 and demonstrate an application of the formalism to a complete clinical guideline in Sect. 5. Section 6 discusses the findings and clinical relevance. Conclusions are found in Sect. 7.

2 Background and Related Work

CPGs are typically formalized as task-network models [5], i.e. hierarchical plans that comprise constructs for decisions and actions as well as embedded sub-plans. To enable automated execution, decisions are generally modeled as decision trees or tables [9], with data processing specified using expression languages (e.g. GELLO) [5].

Regardless of the specific formalism, task-network models encapsulate the control flow (i.e. the logical ordering) between different tasks over time [5]. As a result, they assume a centralized system architecture in which a supervisory component controls the execution of guidelines (i.e. the application of guidelines to patient data). To reduce reliance on such components, Shalom et al. in 2015 proposed a projection mechanism whereby self-contained portions of a clinical guideline are identified for execution in parallel with the overall plan [10]. This mechanism was implemented in the MobiGuide patient guidance system and was demonstrated in the atrial fibrillation and gestational diabetes domains [6]. The MobiGuide system contains two decision support systems: a front-end system running on the patient's smartphone to execute guideline fragments locally; and a back-end system running on hospital servers to execute the overall guideline and "project" the appropriate portions to recipient devices [10].

While this projection mechanism can be extended to an arbitrary number of "local" devices, it still requires a supervisory component to project guideline fragments. To remove the reliance on a centralized controller, formalisms based on production rules may be adopted instead, the main exemplars of which include the Arden Syntax [8] and the OpenEHR Guideline Definition Language [11]. In general, these rules encapsulate the data flow of clinical guidelines, thus they do not exhibit control flow dependencies and can therefore be executed in parallel. However, they also do not intrinsically support the distinction between different types of tasks featured in clinical guidelines, such as diagnosing a patient's condition and making a therapeutic decision.

Therefore, while our formalism also focuses on data flow rather than control flow, we base our formalism on previous work [3] which introduced an informal conceptual data flow model of disease management. Our model comprises four types of data flow processes interacting with the environment (e.g. the patient):

- Monitoring (M), the process of making observations about the patient.
- Analysis (A), the process of making assessments about the state of the patient.
- Decision (D), the process of deciding on the appropriate therapeutic plan.
- Effectuation (E), the process of executing the decided plan, which may involve performing an action or controlling the execution of a process (possibly itself).

Our MADE model (Fig. 1) was demonstrated to be a useful conceptual tool for analyzing clinical guidelines and designing decision support systems in the context of pervasive healthcare [3]. To support interoperability in pervasive healthcare systems, we also derived from it a reference information model (the MADE RIM) for representing clinical data [2]. Building on this work, we present in this paper a formalism for representing guideline knowledge as interconnected MADE processes.

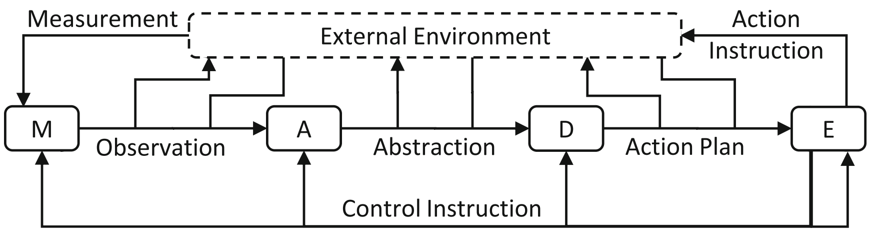

Fig. 1. The MADE model for disease management.

3 The MADE Guideline Formalism

3.1 Overarching Model of Clinical Guidelines

Our MADE formalism is specified using (axiomatic) set theory and comprises 26 set definitions, 13 function signatures and 34 logical invariants. This paper focuses on the set definitions, which specify the data types that constitute our formalism. In particular, we abstract away the technical aspects of pervasive healthcare systems and model a clinical guideline as a set of processes that are fully connected with each other, with data flowing instantaneously between them.

In turn, each process is modeled to comprise four components: a unique ID, a data state to store previously input data, a control state which determines when the process is activated and a specification of the instructions for the process when it runs. This leads to the following two set definitions:

$$Guideline = \mathcal{P}(Process) \tag{1}$$

$$Process = Id \times DataState \times ControlState \times InstSpec \tag{2}$$

Processes are modeled to execute at each time instant, and during execution each process updates its states and outputs a set containing 0 or 1 data item depending on its states, its input data, the current date-time as well as its specified instructions. In accordance to Fig. 1, six types of data are distinguished, which together constitute the MADE RIM as specified in [2]. Here we present the specification of *InstSpec*, of which four types are distinguished, one for each type of MADE process:

$$\begin{aligned} Data = {} & Measurement \cup Observation \cup Abstraction \cup ActionPlan \\ & \cup ActionInstruction \cup ControlInstruction \end{aligned} \tag{3}$$

$$InstSpec = MSpec \cup ASpec \cup DSpec \cup ESpec \tag{4}$$

For ease of understanding, the behavior of each type of MADE process is presented in this paper using natural language English, although it is in fact specified using function signatures and logical invariants. For example, the execution of a process as described above can be formally specified as the following function:

$$execute : Process \times \mathcal{P}(Data) \times DateTime \rightarrow Process \times \mathcal{P}(Data), \text{ such that} \tag{5}$$

$$\begin{aligned} & \forall p \in Process,\ d_{in} \in \mathcal{P}(Data),\ t \in DateTime. \\ & (\neg isProcessActivated(\pi_3(p), t) \Rightarrow d_{out} = \{\}) \wedge |d_{out}| \leq 1 \wedge \\ & \pi_1(p) = \pi_1(p_{out}) \wedge \pi_4(p) = \pi_4(p_{out}) \end{aligned} \tag{6}$$

Here $\pi_i(x)$ refers to the i^{th} element of x, (p_{out}, d_{out}) is the result of $execute(p, d_{in}, t)$ and *isProcessActivated* is a function that determines whether the process is activated or not given its control state and current date-time. Thus, Eq. 6, which is a logical invariant, specifies that a process may only output data when dictated by its control state and that at most 1 data item can be generated. If the process is not activated, it can still update its data state and control state, but under no circumstances can it be transformed into another process by changing its identifier or its instructions.

3.2 Model of Monitoring Processes

Two types of Monitoring processes are distinguished to reflect the two types of observations specified in [2], viz. observed properties and observed events. In our formalism, observed properties are generated by performing digital signal processing, e.g. noise filtering, on input measurements. Thus for Monitoring processes which output observed properties, the specification comprises:

- A time window indicating the duration beyond which data is considered irrelevant.
- A mathematical function that operates on the measurements that have been filtered using the time window, returning the value of the output property.
- An output type identifying the specific type of observed property to generate.

$$\begin{aligned} \mathit{MSpec} &= \mathit{PropertySpec} \cup \mathit{EventSpec}, \text{ where} \quad (7)\\ \mathit{PropertySpec} &= \mathit{TimeWindow} \times \mathit{ValueFunction} \times \mathit{OutputType} \quad (8)\\ \mathit{TimeWindow} &= \mathit{Duration} \quad (9)\\ \mathit{ValueFunction} &= \mathcal{P}(\mathit{Measurement}) \rightarrow \mathit{PropertyValue} \quad (10)\\ \mathit{OutputType} &= \mathit{Id} \quad (11) \end{aligned}$$

Whenever a Monitoring process for observed properties is activated, all input data (including those stored in its data state) are filtered using the time window to remove data that are either not measurements or not relevant at the current date-time. The process then feeds the remaining measurements into its value function and outputs an observed property with the specified output type and computed value.

Unlike observed properties, observed events can exhibit a start and an end, and they can only be assigned a boolean value to indicate whether they occurred or not [2]. Thus Monitoring processes for observed events are specified to comprise:

- A time window and predicate specifying the conditions indicating the event's start.
- A time window and predicate specifying the conditions indicating the event's end.
- An output type identifying the specific type of observed event to output.

$$\begin{aligned} \mathit{EventSpec} &= \mathit{EventTrigger} \times \mathit{EventTrigger} \times \mathit{OutputType}, \text{ where} \quad (12)\\ \mathit{EventTrigger} &= \mathit{TimeWindow} \times \mathit{TriggerPredicate} \quad (13)\\ \mathit{TriggerPredicate} &= \mathcal{P}(Measurement) \rightarrow \mathit{Boolean} \quad (14) \end{aligned}$$

As with the time window in *PropertySpec*, the time windows in *EventSpec* are used to filter out irrelevant measurements; the remaining measurements are input into the corresponding predicates to determine if the start or end of the event is detected. If the start is detected, the Monitoring process will then search through the historical data to determine when the event ended; during this time period, the event did not happen and therefore its value would be false. Similarly, if the end is detected, the process will search for the date-time starting from which the event occurred.

3.3 Model of Analysis Processes

To generate abstractions from observations, Analysis processes comprise:

- An output type identifying the specific type of abstraction to output.
- A set of abstraction triplets, each containing:
  - A time window for filtering out expired measurements.
  - A predicate on the filtered observations that specifies the condition under which an abstraction should be generated.
  - An abstraction function that accepts a set of observations as input and returns the appropriate value for the output abstraction (if one is generated).

$$\begin{aligned} ASpec = OutputType \times \mathcal{P}(&TimeWindow \times AbstractionPredicate \\ &\times AbstractionFunction), \text{ where} \qquad (15) \\ AbstractionPredicate &= \mathcal{P}(Observation) \rightarrow Boolean \qquad (16) \\ AbstractionFunction &= \mathcal{P}(Observation) \rightarrow AbstractionValue \qquad (17) \end{aligned}$$

Like observed events, abstractions are valid over a date-time range [2]. However, unlike observed events, abstractions are not generated by detecting start and end conditions. Whenever an Analysis process is activated, it iterates through its abstraction triplets, and for each triplet it applies the abstraction predicate and abstraction function to the data that has been filtered through the corresponding window. If a predicate is satisfied, the iteration terminates and the process outputs an abstraction with the specified type and corresponding value—otherwise, the process does not generate any abstraction. The valid date-time range of the output abstraction is computed by finding the longest period (starting from the current date-time) during which the predicate remains satisfied and the computed value remains unchanged.

3.4 Model of Decision Processes

Decision processes comprise a plan template from which a new action plan can be instantiated as well as a set of decision criteria governing the conditions under which that action plan should be enacted. In our formalism, decision tables are adopted for decision-making, such that the decision criteria are specified as predicates over the input abstractions. Furthermore, reflecting the specification of action plans, plan templates are modeled to comprise a set of instruction templates, of which three types are distinguished, one for each type of scheduled instruction in an action plan [2]:

- Homogeneous action instructions for actions that exhibit a rate and duration.
- Culminating action instructions for actions that exhibit an end goal.
- Control instructions for controlling the execution of MADE processes by setting their schedules and/or status (whether they should be running or paused).

$$DSpec = PlanTemplate \times \mathcal{P}(DecisionCriterion), \text{ where} \tag{18}$$

$$PlanTemplate = PlanType \times \mathcal{P}(ControlTemplate \cup HomogeneousActionTemplate \cup CulminatingActionTemplate) \tag{19}$$

$$DecisionCriterion = \mathcal{P}(Abstraction) \rightarrow Boolean \tag{20}$$

In contrast to Monitoring and Analysis processes, Decision processes do not require a time window; the only relevant abstractions are those that are valid at the current date-time. These abstractions are checked against the decision criteria; if any criterion is triggered, an action plan will be instantiated from the plan template. The schedule of each instruction in the plan is computed from the current date-time and the relative schedule of the corresponding instruction template.

3.5 Model of Effectuation Processes

Since action plans already contain the complete details of all instructions to be executed and when, Effectuation processes simply specify which instructions they are responsible for effectuating. More specifically, Effectuation processes comprise:

- A specification of the target scheduled instructions to be effectuated by the process. Targets are identified by type of action plan, type of target instruction as well as a predicate on the scheduled instruction.
- An output type indicating the specific type of instruction to output.

$$ESpec = \mathcal{P}(TargetScheduledInstruction) \times OutputType, \text{ where} \tag{21}$$

$$TargetScheduledInstruction = PlanType \times InstructionType \times InstructionPredicate \tag{22}$$

$$InstructionPredicate = ScheduledInstruction \rightarrow Boolean \tag{23}$$

When activated, an Effectuation process filters out any action plans that are not valid at the current date-time and extracts, from the remaining action plans, all relevant scheduled instructions based on the specified targets. From these relevant scheduled instructions, the process then determines if any should be effectuated at the current date-time (given their schedule). If yes, then an instruction will be generated with the specified output type, which may possibly be more specific than the scheduled instruction. For example, an action plan may require an hour's (unspecified) exercise daily while an Effectuation process may instantiate this as an instruction to walk on a treadmill.

4 Reference Implementation

To ensure that the formalism is executable and to clarify any ambiguities, a reference implementation of the formalism was developed as a set of libraries on top of the language Rosette. Clinical guidelines can be formalized using this reference implementation into Rosette programs (Fig. 2), which would comprise interconnected MADE processes. Subsequently, by executing those programs, clinical guidelines can be applied onto (simulated) patient data to demonstrate the semantics of the formalism.

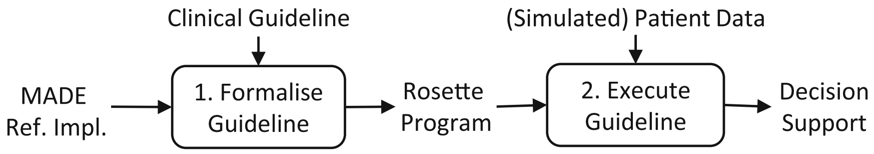

Fig. 2. The overall procedure for using the reference implementation.

The source code is available at https://github.com/nlsfung/MADE-Language. To summarize, each data type in the formalism is implemented as a structure (which is analogous to classes in objected-oriented programs), while behaviors are implemented as procedures. Furthermore, elements of data types are generally implemented as fields in the corresponding structures, with the exception of *InstSpec* which is implemented using interfaces (to ensure that it remains constant over time). Clinical guidelines can then be formalized by extending the structures and implementing the interfaces; during execution, these structures would be instantiated into concrete data items.

For example, a process to analyze ketonuria may be formalized as the following structure. It inherits the data state and control state fields from `analysis-process` and implements the `gen:analysis` interface to return the appropriate output type (`ketonuria`) and abstraction triplets (denoted by `x`).

```
(struct analyze-ketonuria analysis-process ()
#:methods gen:analysis [
(define (analysis-process-output-type self) ketonuria)
(define (analysis-process-output-specification self) x)])
```

Apart from an interpreter for executing programs, Rosette also provides access to off-the-shelf constraint solvers to analyze them [12]. This allowed the reference implementation to be verified against 34 logical invariants derived for the formalism (e.g. Eq. 6). Each invariant was translated into an assertion in Rosette, which was then checked (using a constraint solver) whether a concrete counterexample can be found that violates it. If yes, then the assertion is not valid. Otherwise, as is the case with the 34 invariants, it provides evidence (but not a definitive proof) that the assertion is valid.

5 Case Study: Gestational Diabetes Guideline

The MADE formalism was tested by formalizing a complete clinical guideline for gestational diabetes (GD) [7] that has previously been adopted to evaluate the MobiGuide system [4]. The result was a Rosette program (also available on GitHub) that comprises 0 Monitoring, 4 Analysis, 22 Decision and 29 Effectuation processes. As an example, we focus on the guideline fragment shown in Fig. 3, which relates to the decision to increase the carbohydrates intake of a GD patient. To highlight the identified MADE processes, we annotated the extract by underlining followed by an inserted `[M]` for Monitoring, `[A]` for Analysis and `[D]` for Decision processes.

```
``... The patient measures ... ketones in the urine every day
at fasting conditions [M]. ... The results of ketonuria could
be: a) positive (++); b) positive (+); c) negative (+/-);
d) negative (-); e) negative (--). ... In case of ketonuria
detection (the number of ketonuria measurements with result
``positive'' is equal or higher than 3 in a period of time of
one week) [A]:- If the patient was COMPLIANT with the prescribed
diet [A], the nurse decides to increase the carbohydrates intake
either at dinner or at bedtime ... by 1 unit [D] ... ''
```

Fig. 3. Extract from the GD guideline [7] annotated by [3].

Since urinary ketone levels must be monitored manually by the GD patient, only two processes were formalized for this example. The first is the analysis of ketonuria, which requires a window of 7 days and outputs a ketonuria abstraction with value positive if more than two urinary ketone values are `+` or `++`. The second is the decision to increase carbohydrates intake, which was confirmed by clinicians to be automatable. It accepts as input ketonuria and diet compliance abstractions and outputs an action plan to increase carbohydrates intake at dinner if ketonuria is positive and diet is compliant.

These two processes are specified as follows. In practice, both processes were formalized into Rosette code as with the rest of the GD guideline, all of which was then tested using simulated data (see the Appendix for examples). However, for conciseness, their specifications are presented here using mathematical notation, with constants (denoted using small caps) replacing instances of low-level structures (such as durations).

$$\begin{gathered}
\mathit{AnalyseKetonuria} \subset \mathit{Analysis}, \text{ such that} \\
\forall p \in \mathit{AnalyseKetonuria}.\ \pi_1(p) = \textsc{Analyse Ketonuria} \wedge \\
\pi_1(\pi_4(p)) = \textsc{Ketonuria} \wedge \pi_2(\pi_4(p)) = \{(\textsc{One Week}, \\
d_{in} \mapsto |\{d \mid d \in d_{in} \wedge d \in \mathit{UrinaryKetoneLevel} \wedge \pi_4(d) \in \{+, ++\}\}| \geq 3, \\
d_{in} \mapsto \textsc{Positive})\}
\end{gathered} \tag{24}$$

$$\begin{aligned}
&\mathit{DecideIncreaseCarbohydrates} \subset \mathit{Decision}, \text{ such that} \\
&\qquad \forall p \in \mathit{DecideIncreaseCarbohydrates}. \\
&\pi_1(p) = \textsc{Decide Increase Carbohydrates} \wedge \\
&\pi_1(\pi_4(p)) = (\textsc{Dietary Plan}, \{\textsc{Increase Dinner Carb. Intake}\}) \wedge \\
&\pi_2(\pi_4(p)) = \{d_{in} \mapsto (\exists d \in d_{in}.\ d \in \mathit{Ketonuria} \wedge \pi_4(d) = \textsc{Positive}) \wedge \\
&\qquad (\exists d \in d_{in}.\ d \in \mathit{DietCompliance} \wedge \pi_4(d) = \textsc{Compliant})\}
\end{aligned} \tag{25}$$

6 Discussion

6.1 Expressiveness of the Formalism

Experience gained from the case study showed that the MADE formalism has sufficient expressiveness to represent automatable portions of clinical guidelines. In the future, we plan to evaluate this further by, for example, comparing against existing guideline formalisms. However, it can already be observed that our formalism does not support partial specifications for tasks that must be manually performed. For example, since urinary ketone levels are measured manually, the guideline does not explicate how these measurements should be processed (see Fig. 3). For this reason, this and all other measurement tasks could not be formalized into Monitoring processes, which we believe would also apply to other clinical guidelines.

Furthermore, our formalism does not support the personalization of clinical guidelines according to individual patient preferences, which is an important usability feature for providing decision support to patients [6]. In our case study for example, dinnertime must be made explicit (e.g. 7 pm) to formalize the decision process to increase carbohydrates intake; individual adjustments to dinnertime can only be made by directly accessing and changing the formalized guideline. However, we believe the MADE formalism provides a solid foundation on which such extensions can be added, and we will continue to evaluate and improve the formalism using different clinical guidelines.

6.2 Clinical Relevance

The GD guideline was developed by a team of expert clinicians, knowledge engineers and researchers [4], and its clinical relevance has been established in patient trials of the MobiGuide system [6]. However, the MobiGuide system comprises a fixed number (viz. 2) of KBSs, while our aim is to support an arbitrary distribution of knowledge. Therefore, in collaboration with clinicians, patients and other stakeholders, we plan to fully evaluate our formalism and its clinical value by implementing and testing "n-ary" guideline-based pervasive healthcare systems for GD and other clinical applications. Such a system would adopt the MADE reference information model [2] to share data between devices (including EHR repositories) and would implement an optimization algorithm to distribute guideline knowledge so as to maximize system resilience.

While our formalism allows parallelization of clinical guidelines at the knowledge level, an equally valid alternative may be to parallelize them at the source code level, such as by adopting research results from the well-established area of high-performance computing. However, we believe our formalism can offer the advantages of increased transparency and amenability to analysis. In particular, we are currently investigating how to apply formal verification not only on the guideline formalism itself, but also on computerized CPGs expressed in the formalism, such as to ensure that mutually exclusive MADE processes would never be activated at the same time.

7 Conclusions

Guideline-based pervasive healthcare systems can extend evidence-based healthcare beyond the traditional healthcare setting. It is all the more crucial then to have demonstrable system resilience, quality of clinical information and correct operational logic in a highly distributed environment. To this end, the MADE formalism was developed to represent clinical guidelines in the context of pervasive healthcare and is specified using axiomatic set theory to avoid ambiguity and to allow formal analysis. In particular, the reference implementation was formally verified against its specification using Rosette.

Due to its mathematical foundation, the MADE formalism also opens up possibilities for formally verifying clinical guidelines, which we currently investigate. Furthermore, while the formalism has been tested by formalizing a clinical guideline, its clinical relevance must be further evaluated, such as by conducting field studies using a fully implemented system. The overall objective is to improve clinical correctness of guidelines and safety of their implementations as computerized CPGs.

Appendix

The complete formalized guideline for GD was tested by applying it onto simulated patient data. For example, Fig. 4 shows some urinary ketone levels that may be used to test the analysis of ketonuria for GD patients (*AnalyseKetonuria*), which is specified in Eq. 24. Here, the urinary ketone levels are positive from time points 3 to 6, and the Analysis process is activated at time point 7. Each time point is separated by one day, thus at time point 7, there are three or more urinary ketone levels in the past 7 days. Therefore, as expected, a ketonuria abstraction is generated with value positive. This abstraction is valid until time point 11; beyond this point, there are less than 3 positive urinary ketone levels in a 7-day window given the available data.

As another example, Fig. 5 shows a schematic of the data that may be used to test the decision to increase carbohydrates intake (*DecideIncreaseCarbohydrates*), which is specified in Eq. 25. Here, ketonuria is positive from time points 1 to 5 while diet is compliant at time point 2 and from time points 5 to 7. Furthermore, the decision process is activated at even time

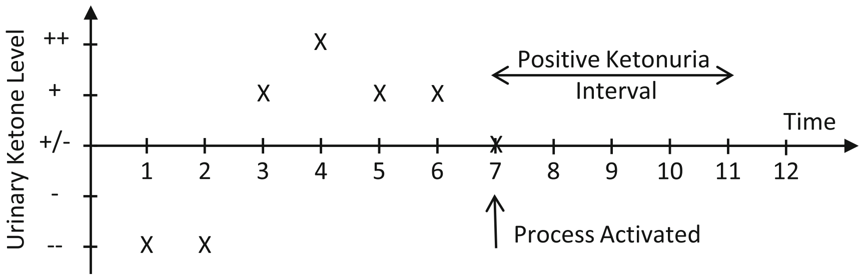

Fig. 4. Example data for testing the analysis of positive ketonuria.

points only, thus on execution, the process outputs an action plan at time point 2 only and not at time point 5, which is as expected (Inv. 6).

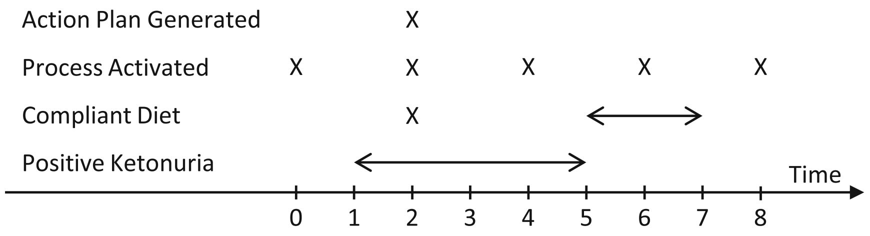

Fig. 5. Example data for testing the decision to increase carbohydrates intake.

References

1. Bardram, J.E.: Pervasive healthcare as a scientific discipline. Method Inform. Med. **47**, 178–185 (2008)
2. Fung, N.L.S., Jones, V.M., Hermens, H.J.: The MADE reference information model for interoperable pervasive telemedicine systems. Method Inform. Med. **56**(2), 180–186 (2017)
3. Fung, N.L.S., Jones, V.M., Widya, I., Broens, T.H.F., Larburu, N., et al.: The conceptual MADE framework for pervasive and knowledge-based decision support in telemedicine. Int. J. Knowl. Syst. Sci. **7**(1), 25–39 (2016)
4. García-Sáez, G., Rigla, M., Martínez-Sarriegui, I., Shalom, E., Peleg, M., et al.: Patient-oriented computerized clinical guidelines for mobile decision support in gestational diabetes. J. Diabetes Sci. Technol. **8**, 238–246 (2014)
5. Peleg, M., Tu, S., Bury, J., Ciccarese, P., Fox, J., et al.: Comparing computer-interpretable guideline models: a case-study approach. J. Am. Med. Inform. Assoc. **10**, 52–68 (2003)
6. Peleg, M., Shahar, Y., Quaglini, S., Fux, A., García-Sáez, G., et al.: MobiGuide: a personalized and patient-centric decision-support system and its evaluation in the atrial fibrillation and gestational diabetes domains. User Model User-adapt Interact. **27**(2), 159–213 (2017)

7. Rigla, M., Tirado, R., Caixàs, A., Pons, B., Costa, J.: Gestational diabetes guideline CSPT. Technical report, MobiGuide Project (FP7-287811) (2013). Version 1.0, 12/02/2013. Internal document
8. Samwald, M., Fehre, K., de Bruin, J., Adlassnig, K.P.: The Arden Syntax standard for clinical decision support: experiences and directions. J. Biomed. Inform. **45**(4), 711–718 (2012)
9. Seyfang, A., Miksch, S., Marcos, M.: Combining diagnosis and treatment using ASBRU. Int. J. Med. Inform. **68**, 49–57 (2002)
10. Shalom, E., et al.: Implementation of a distributed guideline-based decision support model within a patient-guidance framework. In: Riaño, D., Lenz, R., Miksch, S., Peleg, M., Reichert, M., ten Teije, A. (eds.) KR4HC 2015. LNCS (LNAI), vol. 9485, pp. 111–125. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-26585-8_8
11. The openEHR Foundation: Guideline Definition Language v2 (GDL2), CDS Release 2.0.0 edn. (2019). https://specifications.openehr.org/releases/CDS/latest/GDL2.html. Accessed 13 July 2020
12. Torlak, E., Bodik, R.: Growing solver-aided languages with Rosette. In: Proceedings Onward! 2013, pp. 135–152. Association for Computing Machinery, New York (2013)