Recursive Representation of Periodicity and Temporal Reasoning*

Luca Anselma
Dipartimento di Informatica, Università di Torino,
Corso Svizzera 185, 10149 Torino, Italy
Phone: +39 011 6706821 – anselma@di.unito.it

Abstract

Representing and reasoning with repeated and periodic events is important in many real-world domains, such as protocol and guideline management. In this set, it is important to give support to complex periodicities, that can involve non-symmetric repetitions, uncertainness, variability, pauses between repetitions, and nested time intervals. Also, in these domains it can be useful to give support to composite events, as well as classes of events (i.e. types of actions) and instances of events (i.e. specific actions). In this paper we propose a general-purpose domain-independent knowledge server dealing with all these issues. In particular, we describe a compact and (hopefully) user-friendly formalism for representing repetition/periodicity temporal constraints that supports arbitrarily nested repetitions as well as possibly imprecise and variable delays between repetitions. Moreover, we define two algorithms for performing consistency checking on knowledge bases of (possibly repeated/periodic) classes and instances of events retaining the efficiency of less expressive approaches.

1. Introduction

The need to represent and reason with time is crucial within many real world domains; in particular, it is useful to be able to manage different kinds of temporal information such as:

- qualitative temporal relations,
- quantitative temporal relations,
- repeated/periodic temporal relations,
- classes and instances of events,
- composite events.

In the context of temporal constraint reasoning, in literature much attention has been devoted to some of these aspects (e.g. see the surveys in [2, 23]). In particular, since the beginning of the eighties, several domain-independent knowledge servers specifically designed to manage temporal information (i.e. *temporal reasoners*) have been proposed. Temporal reasoners are conceived to deal in an efficient and ad-hoc way with different types of temporal constraints: as regards qualitative temporal constraints it is possible to see, e.g., [1]; as regards quantitative temporal constraints see [5]; as regards "mixed" qualitative and quantitative temporal constraints see [9].

On the other hand, not as much attention has been paid for some other aspects, such as repeated and periodic events (for an approach towards these issues see [10]).

Besides these, there are at least two other aspects which have generally received very little attention, despite their usefulness in a plurality of real-world domains, such as planning, workflow management, and protocol and medical guideline management: classes and instances of events, and composite events. Classes and instances of events, for example, may result effective when dealing with planning issues. In fact, one may wish to specify a general plan with generic actions that are constrained by generic temporal constraints and, then, to execute (i.e. *instantiate*) the plan with actual actions and actual timings, maybe several times and in different contexts. In this set, it is possible to look at the generic actions as *classes* of events, and at the actual actions as *instances* of events corresponding to the actions in the plan. Obviously, the instances must follow (i.e. be consistent with) the temporal constraints on the classes of events.

Furthermore, in domains such as clinical therapies, not only classes and instances of events are needed (because therapies may be regarded as plans), but also both periodic and composite events are, since therapeutic actions often have to be repeated at regular times and may be composed by subactions. Consider, for example, the following excerpt from a clinical guideline for the treatment of multiple mieloma:

(Ex. 1) The therapy for multiple mieloma is made by six cycles of 5 days treatment, each one followed by a delay of 23 days (for a total time of 24 weeks). Within each cycle of 5 days, 2 inner

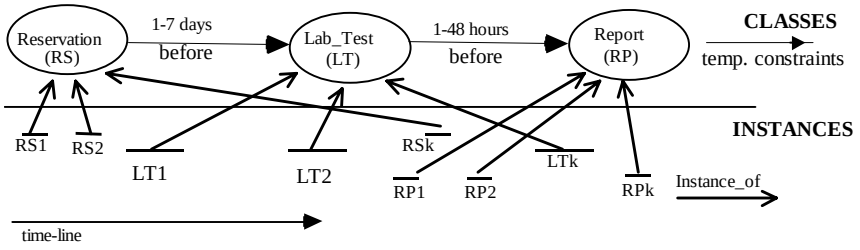
** This is a preprint of a paper accepted for publication in the proceedings of 11th International Symposium on Temporal Representation and Reasoning (TIME'04),*

cycles can be distinguished: the melphalan treatment, to be provided twice a day for each of the 5 days, and the prednisone treatment, to be provided once a day for each of the 5 days. These two treatments must be performed in parallel.
Ex. 1 shows how both periodic and composite events are necessary for the representation of the therapy for multiple mieloma (which we may regard as a composite event) providing treatments (the melphalan treatment and the prednisone treatment) repeated at regular times. Moreover, also qualitative temporal relations are needed, in fact the two treatments must be performed in parallel.
It is crucial to give support to all the above-mentioned kinds of temporal information with a uniform domain-independent approach and hide the complexity of the "machinery" to the user. This goal is motivated by our ongoing work in the domain of guidelines automation ([15, 18, 21, 22], see also subsection 5.2).
A first step in this direction has been made in [20], but with limitations that in some real-world cases it is necessary to overcome: in fact, while that approach can handle Ex. 1, it cannot handle more complex periodicities such as the one sketched in Ex. 2, an excerpt from a clinical guideline for the treatment of Childhood Acute Lymphoblastic Leukaemia:
(Ex. 2) The therapy lasts 88 weeks and it is repeated twice in four years. In the therapy, cotrimoxazole must be given twice daily on two consecutive days every week.
This case presents not only multiple nested time intervals (*four years*, *88 weeks*, *a week*, *a day*), but also temporal constraints between repetitions – the two days must be consecutive –, and uncertainness – it is not specified when the two consecutive days must occur in the week.
In the present work, we make a step further wrt [20]; specifically, we extend the formalism for the specification of repetitions/periodicity to a considerably more powerful one, and we accordingly define new algorithms for temporal reasoning to deal with all the aspects mentioned above. Moreover, we show that, even with the richer representation language, the complexity of the temporal reasoning algorithms does not increase wrt the complexity of the algorithms in [20].
The paper is organized as follows: in section 2, we introduce the problems that our temporal manager has to address; in section 3, we describe the language for representing classes and instances of events, the language for representing constraints on periodicity, and the one employed for internal representation; in section 4, we deal with consistency checking on classes only and with consistency checking on classes+instances by means of constraint inheritance (both with the assumption of complete observability and with the assumption of no observability in the future); moreover, we discuss the complexity of the reasoning mechanisms and present some preliminary experimental results; finally, in section 5, we draw some conclusions, discuss the related works in literature, and present applications and future work.

2. An introduction to the problem

2.1. Temporal constraints about classes+instances

The distinction between classes of events and instances of events is only a specific case of the well-known distinction between classes and instances. Let us show an example:
(Ex. 3) The reservation of a laboratory test (RS) must be done within 1 and 7 days before the laboratory test (LT). The results are reported (RP) within 1 and 48 hours after the end of the test.
Ex. 3 is illustrated in Fig. 1. In Ex. 3, the event of "performing a laboratory test" (*LT*) stands for a class of events, i.e. it stands for a set of individual occurrences (*LT1*, *LT2*, …, *LTk*) of the actual event "performing a laboratory test", that are associated to specific patients and occur in



Correlation:
{<C1,RS1,LT1>,<C2,RS2,LT2>,.., <Ck,RSk,LTk>, <C1,LT1,RP1>, <C2,LT2,RP2>, ..., <Ck,LTk,RPk>}

Fig. 1. A graphical representation for the execution of Ex. 3.

specific intervals of time. These are, of course, instance events of *LT*. The bottom-up arcs in Fig. 1 represent instance-of arcs.
If we wish to check the temporal consistency of the instances of events wrt the classes of events, we have to instantiate the temporal constraints about the classes (in the figure, *1-7 days before* and *1-48 hours before*) on the instances: if the plan states that the reservation *RS* must be done 1-7 days before the lab test *LT*, we have to test whether each instance *RS1*, …, *RSk* is 1-7 days before the *relative* instance *LT1*, …, *LTk*. In doing this, we have to face the problem of relating the instances, which the *correlation* relation (discussed e.g. in [10, 13]) allows us to do (it is no further discussed here because it is out of the scope of this paper).
Moreover, we have to take into account the aspect of *prediction*. The classes of events may have an "existential" role, in fact, the classes of events in Ex. 3 not only impose temporal constraints on the instances, but also "existential" constraints. For example, it would not be possible to have a laboratory test without the relative report. Whenever we test the consistency of the execution of the guideline, we cannot simply inherit the temporal constraints on the correlated instances, but we also have to check whether the proper instances exist.
This problem is closely related with the semantic assumptions we impose on the observations: are the observations complete or is it possible that any instance of event is not observed? As regards future events, are they present in the knowledge base, or (as in the majority of the application domains) is it impossible to know in advance when the future instances of events will occur?
Of course, the semantic assumptions closely influence the possible consistency checking mechanisms, in fact, if the report is not observed and we assume that the observations are not complete, the instances may however be consistent (i.e., the predictive role of the classes is not needed), but if we assume that the observations are complete, a missing report surely leads to an inconsistency.

2.2. Temporal constraints about periodicity/repetition

Repeated events are widespread in many application domains. In many cases (e.g. in the clinical therapies, such as in Ex. 1 and Ex. 2), repeated events are periodic, since they occur at regular times.

As regards the representation of periodicity constraints, we face two alternatives: the *extensional* representation or the *intensional* representation. In the extensional representation, we explicitly represent all the repetitions of each class of events: this could lead in an unnecessary explosion of events, in particular when the number of repetitions is high, such as in Ex. 2. Moreover, if we pursue the aim of hiding the complexity of the time-related tasks to the user, an intensional approach with a compact formalism representing the repetitions may result easier to manage and more "user-friendly". In particular, the formalism has to be powerful (in order to capture the real-world cases), compact, and simple to use (in order to hide the complexity). As regards its expressive power, it has to be able to:

- manage arbitrary nesting of time intervals (e.g., in Ex. 2 we have four nesting levels: four years, 88 weeks, one week, and one day);
- give support for some degree of uncertainty (e.g. in Ex. 2 the guideline does not explicitly constrain the position of the two consecutive days in the week).

In this context, when we impose a repetition pattern, it would be useful to have *periodicity constraints*. These may be considered as a special kind of temporal constraints; to be more specific, while duration constraints are monadic constraints because they regard single instances, and delay/precedence constraints are binary constraints because they regard pairs of instances, periodicity constraints are *n-ary constraints*, because they regard multiple events.
This leads to a further problem that we have to face whenever we want to inherit the constraints from a plan containing repeated classes to instances, that is, we need to associate the instances to the periodicity. If we have constraints like "repeat twice each day A before B", in order to test the precedence relation, we cannot consider all the (instances of) A and all the (instances of) B, but only the pairs of (the instances of) A and B that belong to the same repetition. This may be not a trivial task when the knowledge base of instances only contains *atomic* instances. In the sample constraint, e.g., it is possible that only the occurrences of the instances of A and B are observable, while the "abstract" instance "occurrence of A+B" is not.

2.3. Extending the Integrated Temporal Manager

To summarize, the goal of the work described in this paper is to allow users to abstract from the

above-mentioned problems and to solve them once and for all in a general, domain-independent way, in order to hide the "complexity". To do so, we extend the tractable temporal manager proposed in [20] in the direction of:

- enhancing the expressive power of the representation language, especially the formalism regarding the periodicity/repetition constraints;
- developing new algorithms for temporal reasoning without increasing the complexity of the algorithms in [20].

3. The representation formalism

As regards the representation of the problem, we have followed a layered approach, making a distinction between a high representation level and a low representation level.

The *high level* is the level that interfaces with the user: it is desirable that the language is expressive and that it is possible to provide reasoning facilities. It is important to take into account the trade-off between the expressiveness and the complexity of the reasoning mechanisms. In our approach we have chosen to retain the *tractability*.

The *low level* is meant for the internal representation. We use a "standard" approach, such as the STP one ([5]), where the reasoning mechanism consists in the propagation of constraints.

It is worth noting that, while reasoning on the low level is quite trivial since all-pairs shortest paths algorithms such as Floyd-Warshall's one are correct and complete, filling the gap from the high level to the low level may not be so simple; in fact, one has to face all the problems mentioned in section 2: correlation, association, semantic assumptions, inheritance of periodic patterns, and prediction.

3.1. Representing classes+instances

The high-level constraint language about instances retains the language described in [20]. It provides primitives in order to represent (possibly imprecise) dates, delays between endpoints of events, durations, and qualitative constraints between endpoints of events.

Such language has been deliberately designed in such a way that only constraints that can be mapped onto conjunctions of bounds on differences (i.e., on an STP framework [5]) can be represented (see [16] for more details).

Besides temporal constraints, this high-level language also allows to specify:

- instance-of relations (between an instance and its class),
- correlation relations (between pairs of instances).

Thus, according to our language, the instances are represented with a knowledge base IKB, composed by a quadruple <*IKB_Elements, IKB_Instance_Of, IKB_COR, IKB_Constraints*>, where the first term represents the set of instances, the second one the instance-of relations, the third one the correlation relations, and the last one the temporal constraints between instances.

As regards qualitative and quantitative temporal constraints between classes, we basically retain the simple high-level constraint language of instances. Thus, our language provides primitives in order to describe (possibly imprecise) dates, durations, and delays, as well as *continuous pointizable qualitative temporal constraints* ([25]). Notice, however, that the semantics of such constraints is different depending on whether they apply to instances or to classes (see [16]). Furthermore, the high-level language for classes provides primitives to describe composite events ([20]) and periodicity constraints (see section 3.2).

Thus, classes are represented in the knowledge base CKB, that is a triple <*CKB_Elements, CKB_Part_of, CKB_Constraints*>, where the first term represents the set of classes, the second one represents the part-of relations, and the last one represents temporal constraints (including the constraints on repetition/periodicity).

3.2. Representing repetition/periodicity constraints

The constraints on repetitions and periodicities are temporal constraints of the form

$$Repetition(C, RepSpec),$$

where C is the class to be repeated and *RepSpec* is a parameter that imposes the repetition pattern.

The repetition specification is represented by means of a recursive structure of arbitrary depth,

$$RepSpec = \langle R_1, R_2, \ldots, R_n \rangle,$$

where each level R_i states that the events described in the next level (i.e., R_{i+1}, or – by convention – the class C, if $i=n$) must be repeated a certain number of times in a certain time lapse.

To be more specific, the basic element R_i consists of a triple:

$$R_i = \langle nRepetitions_i, I\text{-}Time_i, repConstraints_i \rangle,$$

where the first term represents the number of times that R_{i+1} must be repeated, the second one represents the time lapse in which the repetitions must be included, and the last one may impose a pattern that the repetitions must follow. We can roughly describe the semantics of a triple R_i as the natural language sentence "*repeat* R_{i+1} $nRepetitions_i$ *times in exactly I-Time*$_i$".
$repConstraints_i$ is a (possibly empty) set of pattern constraints, representing possibly imprecise repetition patterns. Pattern constraints may be of type:

- *fromStart(min, max)*, representing a (possibly imprecise) delay between the start of the I-Time and the beginning of the first repetition;
- *toEnd(min, max)*, representing a (possibly imprecise) delay between the end of the last repetition and the end of the I-Time;
- *inBetweenAll(min, max)* representing the (possibly imprecise) delay between the end of each repetition and the start of the subsequent one;
- *inBetween*((min_1, max_1), …, $(min_{nRepetitionsi-1}, max_{nRepetitionsi-1})$, representing the (possibly imprecise) delays between each repetition and the subsequent one. Please note that any couple (min_i, max_i) may be missing, to indicate that we do not give any temporal constraint between the i-th repetition and the (i+1)-th one.

Let us see an example:
(Ex. 4) Intrathecal methotrexate must be administered 7 times during 88 weeks, never less than 10 weeks apart or more then 14 weeks apart.
Ex. 4 may be represented with a simple one level specification:
Repetition(Intrathecal_methotrexate,
< <7,88wk, {inBetweenAll(10wk, 14wk)}>>)

It is worth noting that neither $repConstraints_i$ nor $nRepetitions_i$ are mandatory. If $repConstraints_i$ is an empty set, the repetitions do not necessarily have to follow any particular pattern. If $nRepetitions_i$ is missing, it is easy to automatically "fill the blank" considering the I-Time of the next level in order to infer the (maximum) number of repetitions that fits in the given I-Time.
Notice that, since we aim at designing tractable algorithms in order to deal with correct and complete consistency checking, we have to impose that I-Times must be specified in an *exact* way.

It should be pointed out that the formalism we are introducing allows to manage different kinds of uncertainty/variability:

- the repetitions are not constrained to completely cover the I-Time, and there may be arbitrary delays between the repetitions;
- the *(min, max)* specifications in $repConstraints_i$ make it possible to indicate variable delays between the repetitions.

$repConstraints_i$ also allows to represent non-symmetric patterns, such as in Ex. 5:
(Ex. 5) Repeat action A every week for 8 weeks on Mondays and Saturdays.
If we regard Sunday as the first day of the week, we represent Ex. 5 as follows:
Repetition(A, <<_, 8wk, ∅>, <2,1wk, {fromStart(1d,1d), toEnd(0d,0d)}> >).

Moreover, the repetitions may be nested at arbitrary depth, representing simple cases with few levels as in Ex. 4 and more complex cases with more levels as in Ex. 2. Ex. 2 may be represented in the following way:
Repetition(Cotrimoxazole, < <2, 4y, ∅>, <_, 88wk, ∅>, <2, 1wk, {inBetweenAll(0,0)}>, <2, 1d, ∅>>),
where the pattern constraint *inBetweenAll(0,0)* in the third triple imposes that the days must be consecutive.

3.3. Internal representation for temporal constraints

As in [20], we model repeated events as composite events and represent the constraints regarding repeated actions into separate STP frameworks, one for each repeated event. Thus, in our approach, the overall set of constraints between classes of events is represented by a tree of STP frameworks (*STP tree* henceforth) ([18]). The root of the tree is the STP which homogeneously represents the constraints between all the classes of events (both composite ones and atomic ones), except repeated events. Each node in the tree is a STP and has a child for each repeated class. Each edge in the tree connects a pair of endpoints in a STP (the starting and ending point of a repeated event) to the STP containing the constraints between its subactions and is labelled with the recursive repetition structure *RepSpec* describing the temporal constraints on the repetitions.

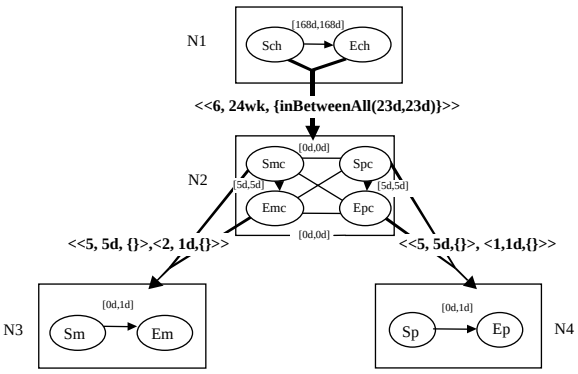


Fig. 2. Graphical representation of the internal representation for Ex. 1.

In Fig. 2, e.g., a graphical representation regarding Ex. 1 is shown.

4. Temporal reasoning

In order to deal with the more powerful formalism described in this work, it is not possible to use the reasoning mechanisms depicted in [20]. In this section, we first describe an algorithm designed for checking the consistency of a knowledge base of (possibly repeated) classes. Then, we describe an algorithm that checks in an integrated way the consistency of instances of events wrt the relative classes, and we show that, despite the more expressive formalism, the complexity of the algorithm does not increase wrt [20].

4.1 Consistency checking on (possibly repeated) classes of events

The procedure *classConsistency* in Fig. 3 tests the consistency of the classes by filling the gap between the high-level expressive representation language and the low-level "simple" internal representation as STP ([5]), making explicit the semantic assumptions carried by the intensional high-level language. At the end of step 2 of the procedure *classConsistency*, *S* is a STP which is semantically equivalent to the STP tree *T*.
This task is accomplished by:

i) visiting recursively the STP tree (task performed by the procedure *unfoldNode*);
ii) "unfolding" the repetitions (task performed by the procedure *unfoldRep*).

The procedure *unfoldNode*, recursively called on each STP node *X* in the STP tree, inserts in *S* (step 1) a class C_X representing the whole node *X*, and (steps 2-3) a class C_A representing each non-repeated class.
For each repeated class C_R (steps 4-7) it calls the procedure *unfoldRep* in order to "unfold" the repetitions. Finally, in steps 8-11 the monadic and binary temporal constraints are carried to *S*.
The procedure *unfoldRep*, after inserting in *S* a class C_1 representing the whole I-Time in which the repetitions must take place, exploits the recursive structure of the repetition specifications to recursively call itself (step 5, please note the shift of the triples) as many times as prescribed by the specification R_1. This is done until the last triple in *RepSpec* is reached (*else* branch of *if* statement in step 4), then the procedure *unfoldNode* is called (step 6) to continue the unfolding on the child node in the STP tree.
Finally, *unfoldRep* adds to the STP (steps 7-12) the constraints corresponding to the semantic assumptions of the construct:
- the repetitions must be included in a time

procedure classConsistency(T : CKB)
(1) initialize S to an empty STP
(2) unfoldNode(root(T), S)
(3) S' := FloydWarshall(S)
(4) **return** S'

procedure unfoldNode(X : STPNode, S : STP)
(1) add to S the placeholder class C_X
(2) **forall** C_A | C_A is not a repeated class in X do
(3) add to S the class C_A
od
(4) **forall** C_R | C_R is a repeated class in X **do**
(5) let RepSpec = (R_1, …, R_n) be the repetition specification of class C_R
(6) C_{sub} := unfoldRep(X, C_R, RepSpec)
(7) add to S the constraints that $C_{sub} \subseteq C_X$
od
(8) **for** each monadic constraint in X **do**
(9) add the constraint to the corresponding classes in S
(10) **for** each binary constraint in X **do**
(11) add the constraint to the corresponding classes in S
return C_X

procedure unfoldRep(X : STPNode, S : STP, C : Class, RepSpec = < R_1, R_2,…, R_n>)
(1) add to S the placeholder class C_1
(2) let R_1 = <$nRep_1$, IT_1, $constrs_1$>
(3) **for** r := 1 **to** $nRep_1$ **do**
(4) **if** R_1 is not the last one in RepSpec **then**
(5) $C_{sub,r}$:= unfoldRep(X, C, (R_2, …, R_n))
else
(6) $C_{sub,r}$:= unfoldNode(child(C, X))
od
(7) add to S the constraint that duration of C_1 is IT_1
(8) add to S the constraints that $C_{sub,i} \subseteq C_1$, i = 1, …, $nRep_1$
(9) add to S the constraints that $C_{sub,i+1}$ is after $C_{sub,i}$, i = 1, …, $nRep_1$-1
(10) add to S the possible constraint fromStart in $constrs_1$ in R_1 between C_1 and $C_{sub,1}$
(11) add to S the possible constraint toEnd in $constrs_1$ in R_1 between $C_{sub,nRep1}$ and C_1
(12) add to S the constraints inBetween and inBetweenAll in $constrs_1$ in R_1 between $C_{sub,i}$ and $C_{sub,i+1}$, i = 1, …, $nRep_1$-1
return C_1

Fig. 3. Algorithms for temporal reasoning on classes of events.

interval lasting exactly IT_1 (step 7);
- each repetition must be included in the I-Time (step 8);
- the repetitions must not overlap (step 9);
- the repetitions must follow the possible pattern in *repConstraints* (steps 10-12).

It is possible to test the consistency of the resulting STP *S* (step 3 of procedure *classConsistency*) by propagating the constraints using the Floyd-Warshall's all-pairs shortest path algorithm.

4.2 Consistency checking on classes+instances

We will start by describing an algorithm that assumes full observability of the events even in the future and total ordering of the instances, and then we will relax the first assumption to include the case where there is no observability in the future. The problem of relaxing the assumption of full observability in the past and the assumption of total ordering is discussed in note 2 and in subsection 5.3. We intend that, whenever an inconsistency is detected, the algorithms report it and stop. For the sake of brevity, we assume that all the input instances are correlated. This is not a restrictive assumption, in fact, since correlation allows to partition instances into independent sets ([10, 13]), the consistency checking of the instances may be iterated for each partition.

In order to test the consistency of classes+instances, the basic idea in the procedure *integratedConsistency* in Fig. 4 is to:

i) test the consistency of the classes and obtain the unfolded STP (step 1);
ii) establish a one-to-one correspondence between the classes and the instances (steps 4-6);
iii) inherit the constraints from the classes to the instances (steps 8-9);
iv) test the consistency of the new "augmented" STP (step 10).

After checking the consistency of the classes (step 1), the instances corresponding to the composite and repeated classes are inserted (step 2) in the IKB[1].

Then, the constraints on the instances are propagated in order to infer a possible not explicit total order between the instances.

Steps 4-6, as said above, try to establish a one-to-one correspondence between the classes in *S* and the instances in *I*. This task may be efficiently performed thanks to the assumption of total ordering on the instances[2].

In the event that an instance that the CKB predicts to be in the IKB is missing (step 6) (e.g. because a repetition is not complete), the procedure stops and reports an inconsistency.

In step 7, we check whether there are instances that are missing the corresponding classes (e.g., because there are more repetitions than expected).

In step 8, monadic constraints (i.e. constraints regarding durations of events) are inherited, and in step 9 binary constraints are inherited, according to the semantics of the constraints on classes ([16]). Steps 8 and 9 may be easily performed thanks to the correspondence between classes in *S* and instances in *I* established in the previous steps.

Finally, consistency checking on the STP on instances, augmented by the inherited constraints, is performed by the Floyd-Warshall's all-to-all shortest path algorithm (step 10).

In the procedure *integratedConsistencyNoFuture* in Fig. 5 we relax the assumption of full observability even in the future to the case where there is no observability in the future. The steps added or changed wrt the procedure *integratedConsistency* are the ones with line numbers in bold type.

The procedure *integratedConsistencyNoFuture*

procedure integratedConsistency(T : CKB, I : IKB)
(1) S := classConsistency(T)
(2) add to I the placeholder instances corresponding to the placeholder classes in S
(3) I' := FloydWarshall(I)
(4) **for** each class C in S taken in temporal order **do**
(5) let i be the first instance of C in I' not yet taken in consideration
(6) if i does not exist **then return** INCONSISTENT
od
(7) **if** there exists an instance in I' not yet considered **then return** INCONSISTENT
(8) inherit the monadic constraints from the classes in S to the corresponding instances in I'
(9) inherit the binary constraints from the classes in S to the corresponding instances in I'
(10)I" := FloydWarshall(I')

Fig. 4. Algorithm for temporal reasoning on classes+instances of events (complete observations).

[1] It is worth noticing that they are not already in the IKB, because we assume that only atomic events may be observed. If this assumption does not hold, then it suffices to simply remove step 2.

[2] To be more specific, without total ordering it could happen that in the IKB there are instances of a repeated class, and we do not know which specific repetitions they belong to. In this case, it would be necessary to perform an inefficient search in order to establish which specific instance corresponds to which specific repetition. Because for each possible correspondence instance-repetition it is necessary to check whether it is consistent with the other temporal constraints, this may lead the problem to intractability. This is the reason why we retain the assumption of total ordering, at least between the instances of a repeated class.

procedure integratedConsistencyNoFuture(T : CKB, I : IKB, **NOW**)
(1) S := classConsistency(T)
(1a)for each instance i in I **do**
(1b) add the constraint the i starts before NOW
(1c)hypothesizedInstances := ∅
(2) add to I the placeholder instances corresponding to the placeholder classes in S
(3) I' := FloydWarshall(I)
(4) **for** each class C in S taken in temporal order **do**
(5) let i be the first instance of C in I' not yet taken in consideration
(6) **if** i does not exist **then**
(6a) add to I a new instance i' of C
(6b) hypothesizedInstances := hypothesizedInstances ∪ i'
od
(7) **if** there exists an instance in I' not yet considered **then** **return** INCONSISTENT
(8) inherit the monadic constraints from the classes in S to the corresponding instances in I'
(9) inherit the binary constraints from the classes in S to the corresponding instances in I'
(10)I" := FloydWarshall(I')
(11)for each i ∈ hypothesizedInstances **do**
if NEC(Start(i) before NOW) **then**
return INCONSISTENT

Fig. 5. Algorithm for temporal reasoning on classes+instances (no observations in the future).

accepts the additional parameter *NOW*, corresponding to the time of the present. In steps 1a-1b we make explicit the fact that it is not possible to observe future events: all observed instances must start before *NOW*.

However, the main differences between the procedures *integratedConsistencyNoFuture* and *integratedConsistency* lie in steps 6-6b and 11: when we do not find an instance that a class predicts to be in the IKB, we no longer report an inconsistency because that instance may start in the future. Thus, there is an inconsistency only in the case that the temporal constraints in IKB and CKB impose that the instance must be observed before *NOW*. Therefore, we collect all the missing instances in the set *hypothesizedInstances* (steps 6 and 6b) and we provisionally insert them in IKB (step 6a). Then, we perform the inheritance and the propagation of the constraints on input+hypothesized instances, and, at the end (step 11), we test whether any hypothesized instance necessarily starts before *NOW*. In this case we report the inconsistency.

It is worth noting that, in the case that the missing instances belong to a repeated class, it is not necessary to hypothesize all the repetitions, but only the first missing one; in fact, if this instance may start in the future, also the subsequent ones will, and it is not necessary to hypothesize them; on the other hand, if this instance must start before *NOW*, we may report the inconsistency even without hypothesizing the others.

4.3 Complexity of the algorithms

As regards the consistency checking on the classes, it is useful to observe that the recursive calls (see step 6 of procedure *unfoldNode* and steps 5 and 6 of procedure *unfoldRep* in Fig. 3), and the *for* loops (see step 4 of procedure *unfoldNode* and step 3 of procedure *unfoldRep*) basically traverse the STP tree, visiting each node as many times as it is repeated. In other words, if there exists a class C such that $Repetition(C, <R_1, R_2, \ldots, R_n>)$, $R_i = <nRepetitions_i, I\text{-}Time_i, repConstraints_i>$ – and C is not a component of another repeated class –, then the class C is visited $\prod_{i=1}^{n} nRepetitions_i$ times. If the class C is a composite class, then, thanks to the recursive call in step 6 of *unfoldRep*, also its component classes are visited $\prod_{i=1}^{n} nRepetitions_i$ times. We accommodate this by expressing the complexity of the algorithm wrt the number C_U of classes in the *extensional* representation, where a class C is present as many times as it is repeated. If we denote with L the number of classes in the intensional representation and with R the maximum number of times that any class is repeated, we can estimate C_U as $C_U = O(R \cdot L)$.

Because the step 7 of the procedure *unfoldNode* is constant in time, and is executed – considering **all** executions – at most $O(C_U)$ times, the procedure is dominated by step 11, that is executed – considering all executions – for every couple of classes, i.e. $O(C_U^2)$ times.

The procedure *unfoldRep* is dominated by steps 8 and 12, that add the constraints imposing that the repetitions must not overlap and the constraints corresponding to the repetition patterns. These steps, considering again all executions, are performed in time $O(C_U)$.

Thus, step 2 of procedure *classConsistency* takes $O(C_U^2)$, and gives as output a STP which contains C_U events and the placeholder classes added by the algorithm. The added classes are at most C_U, so that the STP contains $O(C_U)$ points.

Therefore, since Floyd-Warshall's algorithm is cubic on the number of points, step 3 is executed in time $O(C_U^3)$ and the complexity of *classConsistency* is $O(C_U^3)$.

As regards the consistency checking on instances, we denote with S the number of input instances.

Thanks to a possible precompilation that associates with every class its instances

(performable in O(S)), and thanks to the total ordering of the instances, step 5 of procedure *integratedConsistency* in Fig. 4 is performed in constant time and the entire *for* loop in steps 4-6 is linear in the number of classes. As regards the inheritance of the constraints in steps 8-9, dominates the inheritance of binary constraints, which is quadratic over the number of classes.
Therefore, in the procedure *integratedConsistency* in Fig. 5, steps 1, 3 and 10 dominate, and the overall complexity is O(max{C_U^3, S^3}), or, in terms of the number of input classes L and of the maximum number of repetitions R, O(max{$R^3 L^3$, S^3}).
In the procedure *integratedConsistencyNoFuture*, steps 1a-1c are linear in the number of instances, and steps 6-6b are linear on the number of classes. It is worth noting that for step 11 we may exploit the locality properties of STP constraints proved in [3] and perform step 11 in time linear in the number of instances in *hypothesizedInstances*. The cardinality of *hypothesizedInstances* is at most C_U, since we at most hypothesize one instance for each class. Therefore, even relaxing the hypothesis of complete observability as regards the future, the complexity of integrated reasoning remains O(max{$R^3 L^3$, S^3}).
It should be pointed out that, despite the more powerful representation language wrt the representation language described in [20], the complexity of the reasoning mechanisms does remain the same.

4.3 Preliminary experimental results

The system is currently being developed in Java. We provide preliminary experimental results regarding the algorithm that checks the consistency of the classes only, namely, the procedures *classConsistency*, *unfoldNode* and *unfoldRep* in Fig. 3. The system has been implemented in JDK 1.4. The experiments were run on a PC with a Pentium IV CPU at 2 GHz with 1 GB RAM and Windows 2000 operating system.
The system was provided with five knowledge bases of classes, with increasing number of classes. In Tab. 1 the number of classes C_U (as described in section 4.3) and the time for the consistency checking of the CKBs are reported.
More extensive experiments are needed in order to evaluate the integrated consistency checking on classes+instances.

# of classes (C_U)	Time
10	344 ms
20	360 ms
50	828 ms
100	3203 ms
200	22890 ms

Tab. 1. Number of classes and relative times for checking the consistency of the knowledge base of classes.

5. Conclusions and discussion

In this paper, we describe a formalism for representing temporal constraints on repetition and periodicity in a compact and powerful way. Its intuitiveness makes it easy to use and its recursive structure proves to be adapt to represent arbitrary nested repetitions and supports some degree of uncertainty. We have described two tractable algorithms for consistency checking that address all the aspects mentioned in section 1, namely, classes and instances of events, repetition/periodicity constraints, composite events, and qualitative and quantitative temporal relations. We first have described an algorithm that assumes full observability of the instances of events, and then we have illustrated an algorithm that assumes no observability in the future and full observability in the past.

5.1. Related works

Morris et al. ([10, 11, 12]) dealt only with qualitative constraints between repeated events. Repeated events are used as “classes” of events, with different quantifiers relating them. Morris et al. introduced the notion of consistent scenario in [11] and sketched an algorithm for a scenario consistent with a knowledge base of temporal constraints between repeated events.
Loganantharaj mainly faced the problem of associating possibilistic distributions to qualitative temporal constraints between periodic events ([6]) and to metric constraints concerning the durations of events, which are also expressed using transition rules ([7, 8]). Such constraints are used in a “predictive” way: temporal reasoning is used for projecting the constraints on the durations in the future using the current domain information.
In [19] Terenziani proposed a high-level language to deal with periodicity and in [13, 23] a high-level language to deal with period-dependent qualitative temporal constraints between repeated events, which are used as “classes”. In [14] he also defined an initial algorithm for temporal reasoning with such constraints and a set of instances of events exactly located in time. In [16] he approached the problem of checking the

consistency of classes and instances of events with both qualitative and quantitative constraints; in [17, 18] Terenziani et al. proposed an approach to deal with periodic, qualitative and quantitative constraints between classes of events in clinical guidelines.
Finally, in [20] Terenziani and the author of this paper defined an approach dealing with periodic, qualitative, and quantitative constraints between both classes and instances of events.
This work represents an extension of the work presented in [20]; in particular, our purpose is to improve the representation language described in [20] preserving its efficiency.
To illustrate this, [20] has a much more limited language for the specification of repetitions and periodicity. In fact, that work presents 5 parameters to specify a periodicity: the frame time, the action time, the delay time, the I-Time, and the frequency. The frame time corresponds to the whole time interval in which all the repetitions take place and is subdivided into action times and delay times. Delay times represent fixed delays between an action time and the next one, whereas action times are in their turn subdivided into I-Times, where finally the events occur, at groups of "frequency".
This structure is narrow: in fact, it does not allow to subdivide the intervals into more than three levels (frame times, action/delay times and I-times), thus making it impossible to represent a case such as the one depicted in Ex. 2, which requires – as shown in section 3.2 – four levels.
On the other hand, despite its richer expressive power, the formalism described in this work is more "user-friendly". For example, when dealing with simple cases which do not require multiple levels, the formalism described in [20] implies to arbitrarily impose that action times equal I-Times, whereas the formalism defined in this work allows to simply use fewer recursive levels. To illustrate this, let us suppose that we want to represent the simple case "*repeat A twice for a week*". While with the formalism in [20] it is necessary to state:

FrameTime=1wk, ActionTime=1wk,
DelayTime=0, I-Time=1wk, freq=2,

with the formalism described in this work it is sufficient to state:

$< <2, 1wk, \varnothing> >$.

Moreover, in [20] it is mandatory that the subdivisions are a partition of the higher level; in fact, the union of the action times and the delay times must be equal to the frame time, and the union of the I-Times must be equal to the action time. Furthermore, the "pauses" between the intervals must only be specified at the level of action times, and they must have a fixed duration, which is equal for all the repetitions.
With the periodicity constraint formalism introduced in this work, we provide a more compact and more expressive language. Its recursive structure supports an arbitrary number of nested levels, where any level may or may not be a partition of the higher level: this way we provide for uncertainty in the subdivision of the time intervals. A further support for uncertainty lies in the possibility to specify variable delays between the repetitions.
As regards repetition patterns, not only the repetitions can follow different patterns on each level, but they can also be differently constrained within each level.

5.2. Applications

The need to cope with the various temporal constraints we described in section 1 aroused from our previous work in the field of clinical guideline management. The described system integrates in a joint project with Azienda Ospedaliera S. Giovanni Battista of Torino for the design and development of GLARE (GuideLine Acquisition, Representation and Execution) ([15, 21, 22, 18]). Furthermore, it will integrate in a starting joint project with Cancer Research of London.

5.3. Future work

We are currently trying to extend our approach in order to manage repetitions based on conditions (e.g., while B holds, repeat the action A). This influences both the consistency checking on classes and the consistency checking on instances, because appeals to the predictive role of the classes and therefore deserves specific attention. Furthermore, we are studying the possibility to exploit the locality properties proved in [3, 4] in order to efficiently answer to temporal queries.
Other possible developments comprise the overcoming of some limiting assumptions, such as those of total ordering of the instances and full observability. Although both assumptions are reasonable in the domain of clinical guidelines, there may be domains where they cannot hold.
Unfortunately, these two assumptions make it possible to devise tractable temporal reasoning mechanisms, because it is fundamental to associate an instance with the relative repetition.
In fact, if the two assumptions hold, this task may be performed efficiently (as shown in section 4.2), but, if they do not hold, it would be necessary to

generate a “scenario” for each possible pair (instance, repetition), and test its consistency with the temporal constraints in the knowledge base. Moreover, releasing the tractability for complete reasoning would make it possible to further enrich the expressiveness dealing with different forms of disjunctions of temporal constraints.

In the context of overcoming the limiting assumption discussed above, in order to save some efficiency, we are also investigating the possibility to incorporate the approach described in this work into a backtracking system and to use the temporal constraints in order to restrict the search space.

Acknowledgements

The author wishes to thank prof. Paolo Terenziani for the fundamental contribution as well as his precious help and valuable guidance.

References

[1] J.F. Allen. "Maintaining Knowledge about Temporal Intervals", *Comm. of the ACM*, 26(11): 832-843, 1983.

[2] J. Allen, "Time and Time again: the Many Ways to Represent Time", *Int'l J. Intelligent Systems*, vol. 6, no. 4, pp. 341-355, July 1991.

[3] V. Brusoni, L. Console, and P. Terenziani, "On the computational complexity of querying bounds on differences constraints", *Artificial Intelligence 74(2)*, pp. 367-379, 1995.

[4] L. Console, P. Terenziani, “Efficient processing of queries and assertions about qualitative and quantitative temporal constraints”, *Computational Intelligence* 15(4), pp. 442-465, November 1999.

[5] R. Dechter, I. Meiri, J. Pearl, "Temporal Constraint Networks", *Artificial Intelligence* 49, pp. 61-95, 1991.

[6] R. Loganantharaj, and S. Gimbrone. "Probabilistic Approach for Representing and Reasoning with Repetitive Events", Proc. second International Workshop on Temporal Representation and Reasoning (TIME'95), Melbourne, FL, pp. 26-30, 1995.

[7] R. Loganantharaj, and S. Kurkovsky. "A new model for projecting temporal distance using fuzzy temporal constraints", Proc. IEA/AIE'97, 1997.

[8] R. Loganantharaj, and S. Gimbrone, "Issues on Synchronizing when Propagating Temporal Constraints", Proc. National Conference on Artificial Intelligence Workshop on Spatial and Temporal Reasoning, 1997.

[9] I. Meiri, "Combining Qualitative and Quantitative Constraints in Temporal Reasoning", In Proceedings National Conference on Artificial Intelligence, pp. 260-267, 1991.

[10] R.A. Morris, W.D. Shoaff, and L. Khatib, "Path Consistency in a Network of Non-convex Intervals", *Proc. thirteenth Int'l Joint Conf. on Artificial Intelligence*, pp. 655-660, Chambery, France, 1993.

[11] R.A. Morris, L. Khatib, and G. Ligozat. "Generating Scenarios from specifications of Repeating Events", Proc. second International Workshop on Temporal Representation and Reasoning (TIME'95), Melbourne, FL, pp. 41-48, 1995.

[12] R.A. Morris, W.D. Shoaff, and L. Khatib, "Domain Independent Temporal Reasoning with Recurring Events", Computational Intelligence 12(3) pp. 450-477, 1996.

[13] P. Terenziani, “Integrating calendar-dates and qualitative temporal constraints in the treatment of periodic events”, *IEEE Trans. on Knowledge and Data Engineering 9(5)*, 1997.

[14] P. Terenziani, “Integrated Temporal Reasoning with Periodic Events”, *Computational Intelligence* 16(2), pp. 210-256, May 2000.

[15] P. Terenziani, G. Molino, and M. Torchio, “A modular approach for representing and executing clinical guidelines”, *Artificial Intelligence in Medicine 23*, pp. 249-276, 2001.

[16] P. Terenziani, “Temporal Reasoning with classes and instances of events”, Proc. *TIME 2002*, Manchester, UK, IEEE Press, pp. 100-107, 2002.

[17] P. Terenziani, S. Montani, C. Carlini, G. Molino, M. Torchio, “Supporting physicians in taking decisions in Clinical Guidelines: the GLARE’s ‘what if’ facility”, *Journal of the American Association of Medical Informatics* (JAMIA), Proc. Annual Fall Symposium, 2002.

[18] P. Terenziani, C. Carlini, S. Montani, “Towards a Comprehensive Treatment of Temporal Constraints in Clinical Guidelines”, Proc. *TIME 2002*, Manchester, UK, IEEE Press, pp. 20-27, 2002.

[19] P. Terenziani, “Symbolic User-defined Periodicity in Temporal Relational Databases”, IEEE Transactions on Knowledge and Data Engineering, 15(2), pp. 489-509, March/April 2003.

[20] P. Terenziani, and L. Anselma, “Towards an Integrated Approach Dealing with Part-of, Instance-of and Periodicity Constraints”, Proc. *TIME 2003*, IEEE Society Press, pp. 37-46, 2003

[21] P. Terenziani, S. Montani, M. Torchio, G. Molino, and L. Anselma, “Temporal Consistency Checking in Clinical Guidelines Acquisition and Execution: the GLARE’s Approach”. *Journal of the American Association of Medical Informatics (JAMIA)*, Proc. Annual Fall Symposium, pp. 659-663, 2003.

[22] P. Terenziani, S. Montani, A. Bottrighi, M. Torchio, G. Molino, L. Anselma, and G. Correndo, “Applying Artificial Intelligence to clinical guidelines: the GLARE approach”, Proc. of 8th Congress AI*IA, Lecture Notes in Artificial Intelligence, 2829:536-547, Springer-Verlag, 2003

[23] P. Terenziani, “Towards a Comprehensive Treatment of Temporal Constraints about Periodic Events”, *International Journal of Intelligent Systems*, 18(4), 429-468, 2003.

[24] L.Vila, "A Survey on Temporal Reasoning in Artificial Intelligence", *AI Communications 7(1)*, pp. 4-28, 1994.

[25] M. Vilain, H. Kautz, and P. VanBeek, "Constraint Propagation Algorithms for temporal reasoning: a Revised Report", D.S. Weld, J. deKleer, eds., *Readings in Qualitative Reasoning about Physical Systems*. Morgan Kaufmann, pp. 373-381, 1990.