The Role of Labelled Partitionings for Modelling Periodic Temporal Notions

Hans Jürgen Ohlbach
Institut für Informatik, Universität München
email: ohlbach@lmu.de

Abstract

The key notion for modelling calendar systems as well as many periodic events, for example the seasons, is the notion of a partitioning of the real numbers. *A partitioning of* $\mathbb{R}$ *splits the time axis into a finite or infinite sequence of intervals. Basic time units like seconds, minutes, hours, days, weeks, months, years etc. can all be represented by finite partitionings of* $\mathbb{R}$*. There are a lot of other temporal notions which can be modelled as partitions either: the seasons, the ecclesiastical calendars, financial years, semesters at universities, the sequence of sunrises and sunsets, the sequence of the tides, the sequence of school holidays etc. In this paper a formalization of periodic temporal notions by means of partitionings of* $\mathbb{R}$ *is presented.*

The paper is limited to 4 pages. Therefore it contains only basic ideas and informal descriptions. The technical details can be found in [9].

1. Motivation and Introduction

The basic time units of calendar systems, years, months, weeks, days etc. are the prototypes of periodic temporal notions. They can be modelled with algorithms mapping the begin and end times of a given year, month etc. to dates on a reference time axis [3]. This is sufficient for translating dates between different calendar systems. Another approach is to introduce an intermediate level between the reference time axis and the time units. Different formalizations of this intermediate level have been proposed. In this paper the intermediate level consists of the mathematical concept of partitionings of the time axis. A partitioning splits the time axis into disjoint parts, and these parts can represent years, months etc. They can also represent other periodic temporal notions, for example seasons, financial years, ecclesiastical calendars, semesters, Olympic years etc. By introducing partitionings as an explicit datatype one can then write algorithms which work for basic time units in the same way as for other periodic temporal notions. The algorithm for computing the n^{th} week of a month can also compute for example the n^{th} week in the summer. Or the query 'when is the next Wednesday?' is answered in the same way as 'when is the next summer holiday?'.

Many periodic temporal notions not only correspond to sequences of partitions, but the partitions are named. For example, the days are named 'Monday', 'Tuesday', etc. The seasons are named 'winter', 'spring' etc. There is usually a finite sequence of names, and the sequence of partitions is named by starting at a particular partition, and repeating the sequence of names as often as necessary. To account for this, the concept of *labelled partitionings* is introduced. With labelled partitionings temporal notions like 'next Wednesday', 'last summer' etc. can be modelled in a uniform way. A built-in label is 'gap'. It can be used to denote a partition which logically does not belong to a given partitioning.

The work on partitionings for modelling periodic temporal notions is part of the WEBCAL-project. WEBCAL [2] is a system for understanding, representing and manipulating complex temporal notions from everyday life. It can represent and manipulate fuzzy time intervals [8] to deal with fuzzy notions like 'early today' or 'in the late 20s'. It can deal with different calendar systems, even with historical sequences of calendar systems. Other components are a specification language for specifying application specific temporal notions like 'my weekend' [6, 7]. A prototype of the WEBCAL system is currently being tested.

2. Relation to Other Work

Periodic temporal notions have been modelled in various ways. For example, in their book, 'Time Granularities', Bettini, Jajodia and Wang [1] introduce 'time granularities' as a generalization of partitionings. The 'granules' in time granularities are like the partitions in partitionings, but there can be gaps between two subsequent granules, and the granules can be non-convex intervals. There is an algebra of time granularities, which allows one to define new time granularities from existing ones in various ways. Other approaches are the formalisms of 'collections' [4] and 'slices' [5]. A *collection* is a structured set of intervals where the order of the collection gives a measure of the structure depth. The

Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME'04)
1530-1311/04 $20.00 © 2004 **IEEE**

slice formalism was introduced in [5] as an alternative to the collection formalism in order to have an underlying evaluation procedure for the symbolic formalism.

There are certain fundamental differences between these approaches and the approach in this paper. The basic mathematical structure used in this paper are just ordinary partitionings, which makes things simpler and easier to understand than granules, collections and slices. We get, however, additional expressivity by labelling the partitions with symbolic labels. One of the labels is 'gap', which corresponds to the gaps between granules. The labels have certain advantages. First of all, they are finite data structures for infinite partitions, which is exploited by certain algorithms. Labellings can be put into a hierarchy of labellings, which permits different views on the same partitioning.

The algebraic operations for constructing new granularities or collections from given ones only approximate reality in most cases. For example, the construction of minutes from seconds usually ignores the leap seconds introduced almost every year. A precise definition of minutes,however, must take into account information from a database of leap seconds. A similar problem occurs when days are constructed from hours. The day in spring when daylight savings time is enabled has only 23 hours, and the day in autumn when it is disabled has 25 hours. This information must also be taken from a database. More extreme is the definition of the ecclesiastical calendar, which is anchored at the date for easter. This date depends on the moon cycle, and must therefore be computed algorithmically. Therefore, the proposal in this paper is to define partitionings not algebraically, but algorithmically. This allows one to take all the irregularities of the real calendar systems into account.

Many periodic temporal notions, for example, school holidays depend on concrete dates in concrete calendar systems. Therefore a two-level approach is necessary. At the first level, the basic temporal notions are defined, which are needed for modelling dates in concrete calendar systems, and then new partitionings can be defined which use the dates for their specification.

There is a further difference to the time granularities of Bettini et al. Granules may have holes, and this can be used, for example to define the notion of 'working days within a month'. This is not directly possible in the partitioning approach. Instead, one can define such a notion as a function which maps intervals (one month, for example) to intervals (the working days in the month). These functions may use the partitionings, but their specification and use is a completely different issue and not discussed here [6, 10].

3. Partitionings

A partitioning of the real numbers $\mathbb{R}$ may be for example $(..., [-100, 0[, [0, 100[, [100, 101[, [101, 500[, ...)$. The intervals in the partitionings considered in this paper need not be of the same length (because time units like years are not of the same length either). The intervals can, however, be enumerated by natural numbers (their *coordinates*). For example, we could have the following enumeration

...	$[-100\ 0[$	$[0\ 100[$	$[100\ 101[$	$[101\ 500[$	...
...	-1	0	1	2	...

The formal definition for partitionings of $\mathbb{R}$ which is used in this paper is:

Definition 3.1 (Labelled Partitionings) *A partitioning P of the real numbers $\mathbb{R}$ is a sequence*

$$\ldots [t_{-1}, t_0[, [t_0, t_1[, [t_1, t_2[, \ldots$$

of half open intervals in $\mathbb{R}$ with integer boundaries, such that either the sequence is infinite at one or both ends, or it is preceded by an infinite interval $]-\infty, t[$ (the start partition*) or/and it is ended by an infinite interval $[t, +\infty[$ (the* end partition*).*

A coordinate mapping *of a partitioning P is a bijective mapping between the intervals in P and a subset of the integers. Since we always use one single coordinate mapping for a partitioning P, we can just use P itself to indicate the mapping.*

Therefore let p^P be the coordinate *of the partition p in P.*

For a coordinate i let i^P be the partition which corresponds to i.

A Labelling *L is a finite sequence of strings $l_0, \ldots, l_{n-1}$.*

A labelling $L = l_0, \ldots, l_{n-1}$ is turned into a labelling function *$L_P(p)$ for a partitioning P as follows:*

$$L_P(p) \stackrel{\text{def}}{=} \begin{cases} l_{p^P \bmod n} & \text{if } p \text{ is finite} \\ \text{'startPartition'} & \text{if } p = [-\infty, t[\\ \text{'endPartition'} & \text{if } p = [t, +\infty[\end{cases}$$

where p is a partition in P. ■

In contrast to coordinate mappings, where only one coordinate mapping per partitioning is considered, it makes sense to allow many different labellings for the same partitioning. In most cases the labels are not arbitrary strings, but words of a particular language, and these words have a certain meaning. Therefore one can allow different labellings where the labels are words in different languages. For example the days can be labelled in English: 'Thursday', 'Friday' etc. and in German: 'Donnerstag', 'Freitag' etc.

Labellings can be ordered hierarchically. For example there may be labellings 'BavarianHolidays' and 'BerlinHolidays', which correspond to the different school holiday periods. Both labellings can be a 'sub-labelling' of a labelling 'schoolHolidays'. The difference between 'BavarianHolidays' and 'schoolHolidays' is that the labels in 'BavarianHolidays', which are not the label 'gap', are all labelled 'holiday' in the labelling 'schoolHolidays'. This offers for example the possibility to intersect a partitioning

labelled 'BavarianHolidays' with another partitioning labelled 'BerlinHolidays'. To do this, the labellings 'BavarianHolidays' and 'BerlinHolidays' are replaced by 'schoolHolidays' in both cases, and the intersection of the partitions with label 'holiday' is computed. This is a non-trivial operation which is beyond the scope of this paper. The details of the algebra of labelled partitionings have not yet been investigated.

4. Specifying Partitionings

Five different ways to specify partitionings are presented. Each way influences the details of the functions which map coordinates to partitions and vice versa.

Standard Partitionings are specified by (i) an average length of a partition, (ii) an offset for the partition with coordinate 0, and (iii) a correction function. The correction function computes for a partition with coordinate n the difference between the reference time of the beginning of the partition with coordinate n according to the average length, and the real beginning of the partition with coordinate n. All basic time units, seconds, minutes etc. are modelled as standard partitionings. Other periodic temporal notions which can be modelled by standard partitionings are for example sunrises and sunsets, moon phases, the church year which starts with Easter, etc.

Regular Partitionings are characterized by an anchor date and a sequence of shifts in a given time unit. A typical example is the notion of a semester at a university. In the Munich case, the dates could be: anchor date: 2000/10 (October 2000). The shifts are: 6 months (with label 'winter semester') and 6 months (with label 'summer semester'). This defines a partitioning which extends into the future and the past. The partition with coordinate 0 is the winter semester 2000/2001.

Further examples for periodic temporal notions which can be encoded as regular partitionings are decades, centuries, the British financial year which starts April 1^{st}, the dates of a particular lecture which is every week at the same time.

Duration Partitionings: Regular partitionings are a special case where the shifts are specified in terms of the same partitioning. A natural generalization of this would be to specify the shifts by 'durations'. A *duration* is a list of pairs (number, partitioning). For example (3 month), (1 week) is a duration. A duration partitioning is specified by an achor date and a sequence of durations. For example, we could start with the anchor date 2000 and then define partitions by the three durations (3 month, 1 week), (5 weeks, 10 days), (1 year).

The above notion of a duration is a mathematical concept worthwhile to be investigated separately. It can also play a role in temporal constraint networks [11] if the constraints for temporal distances is not specified in constant time units like seconds, but in time units like months, whose length depends on the position at the time axis.

Calendrical Partitionings are specified by an anchor date and then a finite sequence of partial dates. For example, the seasons can be defined with the anchor date 2000/3/21 (spring begin). The partial dates are 6/21 (summer) 9/23 (autumn) 12/21 (winter) +1/3/21 (spring again). This also defines an infinite partitioning.

Another example could be a whole timetable for a semester in the university, where it is sufficient to specify the dates for one week, and then it is repeated every week.

Date Partitionings: In this version we provide the boundaries of the partitions by concrete dates. Therefore the partitioning can only cover a finite part of the time line.

An example could be the dates of the Time conferences: 1994/5/4 Time94 1994/5/4 gap 1995/4/26 Time95 1995/4/26 . . . 2004/7/1 Time04 2004/7/3.

5. Operations with Partitionings

`shift`: Notions like 'in two weeks time' or 'three years from now' etc. denote time shifts. They can be realized by a function which maps a time point t to a time point t' such that $t' - t$ is just the required distance of 'two weeks' or 'three years' etc. A function $\texttt{shift}(t, m, P)$ is defined where t is a reference time point, n is a real number, and P is a partitioning. The function shifts the time point t by m partitions of the partitioning P.

The definition of `shift` depends on the type of partitioning. The idea for standard partitionings is as follows: if for example the partitioning represents months, and t is the reference time of the current moment in time, then $\texttt{shift}(t, 1, month)$ should calculate the reference time of 1 month in the future. The first problem is that months have different lengths. For example, if t is in February, does 'in one months time' mean in 28 days or in 31 days? This problem can be overcome and a very intuitive solution is obtained if the calculations are not done on the level of reference time points, but on the level of dates. If, for example, t represents 2004/1/15, then 'in one month time' usually means 2004/2/15. That means the reference time must be turned into a date, the date must be manipulated, and then the manipulated date is turned back into a reference time. This is quite straight forward if the partitioning represents a basic time unit of a calendar system (year, month, week, day etc.), and this calendar system has a date format where the time unit occurs.

The next problem is to deal with fractional shifts. How can one implement, say, 'in 3.5 months' time'? The idea is as follows: suppose the date format is year/month/day/hour/minute/second, and the reference time corresponds to, say, 2004/1/20/10/5/1. First we make a shift by three months and we end up at 2004/4/20/10/5/1. This is a day in May. From the date format we take the information that the next finer grained time unit is 'day'. May has 31 days. $0.5 * 31 = 15.5$. Therefore we need to shift the date first by 15 days, and we end up at 2004/6/4/10/5/1. There is still a remaining shift of half a day. The next finer grained time unit is hour. One day has 24 hours. $0.5 * 24 = 12$. Thus, the last date is shifted by 12 hours, and the final date is now 2004/6/4/22/5/1. This is turned back into a reference time.

`advanceCoordinate`$(i, n, P, skipGap)$ advances the coordinate i of a partitioning P by n partitions. If $skipGap = false$ or there is no labelling defined for the partition, then the result is just $i + n$. If $skipGap = true$ then all partitions labelled 'gap' are ignored. The result is $i + n'$ where $n' - n$ is the number of ignored gap partitions. The `advanceCoordinate` function realizes notions like 'tomorrow' or 'in two weeks time' or 'last semester' etc.

`nextCoordinateWithLabel`: this function computes for a given coordinate the coordinate of the next/previous m^{th} partition with the given label. The label of the partition with the given coordinate is ignored. The function can be used to realize notions like 'next Wednesday' or 'Wednesday in three weeks' or 'last summer' or 'the summer 10 years ago' etc.

If the partitionings become part of a calendar system, and concrete time intervals are introduced independently of partitionings, then functions are possible which let partitionings operate on intervals. For example 'new year' can be defined as a function mapping an arbitrary interval I to the first day partition of the year partitions contained in I (see [6, 7]).

6. Summary

The mathematical tool of partitionings, augmented with labels, together with suitable operations, is presented as a useful tool for unifying the treatment of many different periodic temporal notions. This ranges from the basic time units in calendar systems until concrete timetables which are defined for a certain period, usually a week, and then get repeated every week. The characteristics of the concrete partitioning is encoded in the two functions which map coordinates to partitions and vice vers. All other operations are generic. I proposed five different ways to specify a partitioning and gave examples of the kind of periodic temporal notions which can be encoded this way.

References

[1] Claudio Bettini, Sushil Jajodia, and Sean X. Wang. *Time Granularities in Databases, Data Mining and Temporal Reasoning*. Springer Verlag, 2000.

[2] François Bry, Bernhard Lorenz, Hans Jürgen Ohlbach, and Stephanie Spranger. On reasoning on time and location on the web. In N. Henze F. Bry and J. Malusyński, editors, *Principles and Practice of Semantic Web Reasoning*, volume 2901 of *LNCS*, pages 69–83. Springer Verlag, 2003.

[3] Nachum Dershowitz and Edward M. Reingold. *Calendrical Calculations*. Cambridge University Press, 1997.

[4] B. Leban, D. Mcdonald, and D.Foster. A representation for collections of temporal intervals. In *Proc. of the American National Conference on Artificial Intelligence (AAAI)*, pages 367–371. Morgan Kaufmann, Los Altos, CA, 1986.

[5] M. Niezette and J. Stevenne. An efficient symbolic representation of periodic time. In *Proc. of the first International Conference on Information and Knowledge Management*, volume 752 of *Lecture Notes in Computer Science*, pages 161–169. Springer Verlag, 1993.

[6] Hans Jürgen Ohlbach. About real time, calendar systems and temporal notions. In H. Barringer and D. Gabbay, editors, *Advances in Temporal Logic*, pages 319–338. Kluwer Academic Publishers, 2000.

[7] Hans Jürgen Ohlbach. Calendar logic. In I. Hodkinson D.M. Gabbay and M. Reynolds, editors, *Temporal Logic: Mathematical Foundations and Computational Aspec ts*, pages 489–586. Oxford University Press, 2000.

[8] Hans Jürgen Ohlbach. Fuzzy time intervals and relations – the FuTIRe library. Technical report, Inst. f. Informatik, LMU München, 2004. See http://www.pms.informatik.uni-muenchen.de/mitarbeiter/ohlbach/systems/FuTIRe.

[9] Hans Jürgen Ohlbach. The role of labelled partitionings for modelling periodic temporal notions. httpd://www.informatik.uni-muenchen.de/mitarbeiter/ohlbach/homepage/publications-/PRP/abstracts.shtml, 2004. to be published.

[10] Hans Jürgen Ohlbach and Dov Gabbay. Calendar logic. *Journal of Applied Non-Classical Logics*, 8(4), 1998.

[11] L. Vila and L. Godó. On fuzzy temporal constraint networks. *Mathware and Soft Computing*, 1994.