Periodic Temporal Notions as 'Tree Partitionings'

Hans Jürgen Ohlbach

Institut für Informatik, Universität München
E-mail: ohlbach@lmu.de

Abstract. The key notion for modelling calendar systems and many other periodic temporal notion is the mathematical concept of *a partitioning of the real numbers.* A partitioning of $\mathbb{R}$ splits the time axis into a sequence of intervals. Basic time units like seconds, minutes, hours, days, weeks, months, years etc. can all be represented as partitionings of $\mathbb{R}$ with finite partitions. Besides the basic time units in calendar systems, there are a lot of other temporal notions which can be modelled as partitions: 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.
Almost all systems for modelling periodic temporal notions developed so far identify partitions, granules or whatever they are called, by labels or coordinates which are essentially sequences of integers. In this paper it is show how these integer coordinates as identifiers for partitions can be generalised to 'partition access specifiers'. An example for a non-trivial partition access specifier is a path in a tree which represents hierarchically nested partitions. A bus timetable, for example, can be specified this way: '(in very winter, in every week, (in day 0-4, hour 5, minute 20, bus B1, hour 6, minute 20 bus B2 ...), (in day 5-6, hour 8, minute 20 bus B1, ...)), (in every spring ...)...'. A particular 'partition access specifiers' is then a sequence of integers '10/1/2/..'. The first integer represents an absolute coordinate (season 10). The other integers represent shifts: 1 week after the start of season 10, 2 days after the start of this week etc.
The main data structures and algorithms for these 'tree partitionings' are presented in this paper.

1 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 [5]. This is sufficient for translating dates between different calendar systems. Many other periodic temporal notions, however, are very application or user specific, 'my weekend', for example. Therefore a small but active research community is investigating ways to specify periodic temporal notions symbolically. One approach is to introduce an intermediate level between the reference time axis and the time units. Different formalisations of this intermediate level have been proposed. The simplest and most obvious way is to model periodic temporal notions as partitionings of the time axis. This is the approach which is realised in the PartLib library [14] which is part of the CTTN system (Computational Treatment of Temporal Notions [13]). Periodic temporal notions are modelled with labelled partitionings. The labels are names of the partitions ('Monday', 'Tuesday', ... for example).

In their book, 'Time Granularities', Bettini, Jajodia and Wang [3] introduce 'time granularities' as a generalisation of partitionings of the time axis. 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. By means of a special label 'gap' in the PartLib library, one can realise the same kind of granules as in [3]. 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' [10] and 'slices' [11]. A *collection* is a structured set of intervals where the order of the collection gives a measure of the structure depth. The *slice* formalism was introduced in [11] as an alternative to the collection formalism in order to have an underlying evaluation procedure for the symbolic formalism. A selection of papers about the abundant work in this area is [1, 12, 9, 15, 10, 6, 2, 7, 4, 8, 16].

The main characteristics of the algebraic approach is that new periodical temporal notions are constructed from simpler ones with certain basic operators. If the desired temporal notion is complex, many construction steps are necessary until it is constructed. Each construction step introduces a new complexity in the algorithms that work with these notions. In most of these approaches it is, for example, very complex or even impossible to specify, say a bus schedule. The bus schedule may depend on the season, it distinguishes between work days and weekends, between daytime and nighttime etc.

In this paper a new way is introduced to specify such complicated periodic temporal notions as bus schedules in one go, i.e. with one construction step. The result is a partitioning which has the same application interface as, for example, years. All algorithms working with the standard periodic temporal notions can therefore work with these new partitionings as well.

2 Partitionings

Partitionings of the time axis are infinite mathematical structures. Therefore they are not directly implementable with finite data structures. They must be represented on a computer in a more indirect way. We distinguish three aspects of partitionings of the time axis:

1. the mathematical structure. It serves as the semantics for the more concrete descriptions of these objects;
2. the specification of concrete partitionings. There are different ways to specify them. Each type of specification comes with a mathematical structure that has also a serialised text form which can be stored in files;

3. the implementation. There should be a common interface for all types of partitionings, such that the algorithms working with these partitionings are independent of the specification type. The methods of the partitioning application interface must be automatically compilable from the specification.

2.1 Partitionings as Mathematical Structures

In this paper we consider the following partitionings of the time axis:

Definition 1. *The partitionings of the time axis (real numbers) we consider in this paper meet the following conditions:*

1. *they are isomorphic to the integers (their 'coordinates')*
2. *the partions are half open intervals of the form* $[a, b[$ *with integer boundaries* a *and* b.

It is not by chance that half open intervals are used in this definition. Since the partitions in a partitioning do not overlap, one cannot use closed intervals because the endpoints of the closed intervals would be in two different partitions. Open intervals can not be used either because the infima and suprema of the intervals would then not be in any partition at all. Therefore only half open intervals can be used, either of the type $[a, b[$, or of the type $]a, b]$. In most cases there is no preference for either of the two types, but both types should not be used together. In this paper we therefore use the first type $[a, b[$.

Typical examples of such partitionings are years, months, weeks, days, hours etc., but also semesters, financial years, sequences of holiday periods and working periods, sequences of bus arrival times followed by waiting times in a bus schedule etc.

Many of the application specific partitionings, for example, semesters, make sense only for a finite period of time. It turned out that it is always possible, and conceptually as well as algorithmically easier to extrapolate these finite partitionings to the infinity, such that they are really isomorphic to the integers. The finite 'validity period' can be attached as an extra attribute to the specification of a partitioning. It can be used or ignored by an application.

The partitionings of Def. 1 can be enumerated by their corresponding integer coordinates. We have, for example, year 0, year -1, year +1 etc. It is not necessary that the coordinates of the partitions are aligned. That means, for example, month 0 need not be the first month of year 0.

2.2 Partition Access Specifiers (PAS)

We now generalise the concept of integer coordinates to the more general concept of *partition access specifiers.*

Definition 2 (Partition Access Specifier (PAS)). *A* partition access specifier *is a mathematical structure which is isomorphic to the integers. It is specified by a distinguished element* p_0, *an injective* increment function *and an injective* decrement function.

At first glance this seems to be a very uninteresting idea. But there are mathematical structures which are very different to the integers, but still isomorphic to them. The example, which is relevant for this paper, are paths in an infinite tree with finite depth. If the subnodes of all nodes are ordered, one can enumerate the paths in the same way as the integers. The way we express dates, for example, makes use of such a tree structure. We do not say, this is day number 13230 after day 0, for example. Instead, we say, this is year 2006, third month within this year, 23rd day within this month. Year/month/day specify three levels in a tree where the nodes are the corresponding integers. Nevertheless, it is clear that dates like 2006/3/23 can be used to enumerate the infinite sequence of day coordinates in the same way as ordinary integers.

In this example the distinguished element p_0 is the path 0/0/0. The increment function would be going from the date of one day to the date of the next day. The decrement function would be going to the date of the previous day.

2.3 Specification of Partitionings

A specification of a partitioning must be a mathematical structure from which the isomorphism to the integer coordinates or, more general, to partition access specifiers, can be derived.

In [14] I have presented different types of specifications. The first type of partitionings were called 'algorithmic partitionings'. They were characterised by implementing the isomorphism to integers directly. All the standard periodic temporal notions, years, months, weeks etc., but also Easter time, sun rises, tides etc. are of this type. The implementation can in particular take into account all the nasty and irregular features of real calendar systems, leap years, leap seconds, daylight savings time, time zones etc.

Another type of specification were called 'duration partitionings'. They are specified by giving an anchor date and a list of durations. For example, one can specify semesters in this way. The anchor date could be first of October 2000. The durations could be '6 months' (for the winter semester) and '6 months' (for the summer semester).

Another type are 'date partitionings', which are specified by concrete dates. An example could be the seasons. 2000/3/21 spring 2000/6/21 summer 2000/9/23 autumn 2000/12/21 winter 2001/3/21 specifies the seasons for one year. The extrapolation mechanism extrapolates them to the infinity.

2.4 Implementation of Partitionings

The common interface to all partitioning types consists of

– the abstract class 'Partition Access Specifier' (PAS) with a distinguished element p_0, together with an *increment* and a *decrement* function. Concrete subclasses are the integers, and the 'Tree Partition Access Specifier (TPAS. Def. 5)' which represent paths in a tree;

– a function $PAS_P(t)$ which maps a time point t to the partition access specifier of the P-partition containing t;
– a function $startOfPartition_P(pas)$ which maps a PAS to the start of the P-partition denoted by pas.
– a function $endOfPartition_P(pas)$ which maps a PAS to the end of the P-partition denoted by pas. The default definition is
$endOfPartition_P(pas) = startOfPartition_P(increment(pas))$.
If the *increment* function is expensive, it may, however, be more efficient to implement $endOfPartition_P$ directly.

3 Basic Tree Partitionings

Many practical partitionings of the time axis are not like years, days etc., where all partitions have the same status. In a bus timetable, for example, we want to distinguish the one or two minutes where the bus is at the bus stop from the rest of the time where the bus is not there. Therefore we generalise the simple tree structure of dates to a more general structure. First of all, we need to relate the levels of the tree to (previously defined) partitionings. 'year/month/day' is a simple example. A not so simple example is 'year/week/day'. The three partitionings in the first example are aligned. This is not the case in the 'year/week/day' example because a year and a week do not always begin at the same time. Since in our application it is absolutely necessary to identify the first week in a year, or more general, in a sequence $P_0/\ldots/P_n$ of partitionings to identify the first P_i–partition within a P_{i-1}–partition, we need some extra information. There is a wide range of criteria which can be used to identify the first P_i–partition within a P_{i-1}–partition. Fortunately there are only very few ones which occur in practical applications.

Definition 3 (Inclusion Condition). *Given a sequence $P_0/\ldots/P_n$ of partitionings, the following predicates ('inclusion conditions') can be used to specify which P_i–partition counts as the first one in a P_{i-1}–partition:*

– *'subset': the earliest P_i–partition which is a subset of the P_{i-1}–partition;*
– *'overlaps': the earliest P_i–partition which overlaps with the P_{i-1}–partition;*
– *'bigger_part_inside': the earliest P_i–partition whose bigger part is in the P_{i-1}–partition.*

'subset' and 'bigger_part_inside' are only well defined if the P_{i-1}–partitions are big enough to contain a P_i–partition. In implementations one should issue a warning if this is not the case. 'bigger_part_inside' is actually the condition which is used to determine the first week in a year.

Definition 4 (Partition Access Format (PAF)). *A Partition Access Format (PAF) is a sequence $P_0, I_0/\ldots/P_n, I_n$ of tuples consisting of a partitioning and an inclusion condition.*

The first inclusion condition I_0 is irrelevant and may be omitted. If the P_{i-1}–partitions are aligned with the P_i–partitions, we may omit the inclusion condition I_i as well.

Definition 5 (Tree Partition Access Specifier (TPAS)). *A tree partition access specifier with respect to a partition access format* $P_0, I_0/\ldots/P_n, I_n$ *is a sequence* $k_0 \ldots k_n$ *of integers.* $k_1 \ldots k_n$ *are non-negative.*

A TPAS can be shorter than the corresponding PAF. In order to avoid purely technical case distinctions, we assume in this paper that both have always the same length.

For the bus schedule example (Ex. 1, below) we might have 2006/10/3/5/20 as a TPAS. It indicates the particular partition where the bus is at the bus stop. 2006/10/3/5/21 indicates the *same* partition because the bus stays there 2 minutes.

A Tree Partition Access Specifier has a meaning with respect to a Partition Access Format. Essentially it denotes an ordinary date.

Definition 6 (Semantics of TPAS with respect to PAF).
Let $D \stackrel{\text{def}}{=} P_0, I_0/\ldots/P_n, I_n$ *be a Partition Access Format. Let* $d \stackrel{\text{def}}{=} d_0/\ldots/d_n$ *be a TPAS:*
The interpretation $\Im_D(d)$ *maps the TPAS to a* P_n*–partition as follows:*
Let $J_0 \stackrel{\text{def}}{=} [\mathit{startOfPartition}_{P_0}(d_0), \mathit{endOfPartition}_{P_0}(d_0)[$.
If $n = 0$ *then* $\Im_D(d) = J_0$,
otherwise for $l = 1, \ldots, n$: *let* J'_l *be the leftmost* P_l*–partition in* J_{l-1} *with respect to the inclusion condition* I_l. *Let* J_l *be the* P_l*–partition obtained by moving from* J'_l k P_l*–partitions forward.*
Let $\Im_{D,l}(d) \stackrel{\text{def}}{=} J_l$ *and* $\Im_D(d) \stackrel{\text{def}}{=} J_n$.

This definition is nothing else than the usual date computation. For example, with the PAF year/month/day and the TPAS 2000/5/20, we first compute the year 2000. Within this year we locate the first month and move 5 months forward. Within this month, we locate the first day and move 20 days forward.

One can also go the other way round and compute for a time point t and a Partition Access Format the corresponding TPAS.

Definition 7 ($PAS_D(t)$). *Let* t *be a time point and let* $D \stackrel{\text{def}}{=} P_0, I_0/\ldots/P_n, I_n$ *be a Partition Access Format.*
We define the function $PAS_D(t)$ *as follows:*
Let a *be the start of the* P_0*–partition containing* t *and let* d_0 *be its coordinate*[1].
For $i = 1, \ldots, n$:
Let $d_i \stackrel{\text{def}}{=} k - l$ *where* l *is the coordinate of the* first *(according to the inclusion condition* I_i*)* P_i*–partition within the* P_{i-1}*–partition containing* a*, and* k *is the coordinate of the* P_i*–partition containing* t. *Let* a *now be the start of the* P_i*–partition containing* t.

$PAS_D(t) \stackrel{\text{def}}{=} d_0/\ldots/d_n$.

[1] In most cases the partitionings of a PAF will be partitionings like years etc. with integer coordinates. Therefore we shall continue to speak of 'coordinates' in this context. Nothing changes, however, if the partitionings in the PAF have more general Partition Access Specifiers as their coordinates.

For a PAF 'year/month/day', this algorithm computes the date for a time point t: d_0 is the coordinate of the year. d_1 is the difference between the month coordinate containing t and the month coordinate of the beginning of the year. d_2 is the difference between the day coordinate containing t and the day coordinate of the beginning of the month.

The function $PAS_D(d)$ works even for dates like 2006/0/32 (thirty third of January 2006), although January has only 31 days. PAS_D(2006/0/32) moves from the beginning of January 32 days forward. It ends up at the second of February. This date is in fact equivalent to the more *standard* date 2006/1/1[2].

Sometimes it is useful to be able to also work with these nonstandard PAS (dates). Therefore we define a function *standardise* that standardises a PAS. The definition is informal. The concrete details are technical, but not very exciting. $standardise_D(d_0/\dots/d_n)$ goes from d_n backwards to d_0. If d_n is too large, d_{n-1} is incremented. If $d_n < 0$ then d_{n-1} is decremented etc.

Definition 8 (Standardised PAS). *Given a Partition Access Format D and a Tree Partition Access Specifier d, we define a function $standardise_D(d)$ which turns d into its standardised form.*

3.1 Tree Partitioning Specifications

A Partition Access Format specifies the levels in a 'Partition Access Tree' (PAT), which is the kernel of a Tree Partition Specification (TPS).

Definition 9 (Partition Access Tree (PAT)). *A PAT is a tree whose nodes are labelled either*

- *with a range '$n - m$' of integers ($n \leq m$), or*
- *with * (which stands for the maximally admissible set of integers)*
- *or with '$n - *$' where n is an integer (which stands for all maximally admissible integers $\geq n$),*

such that for each node N,

- *either there is only one subnode with label *, or*
- *the labels of the subnodes denote disjoint sets of integers and are ordered in increasing order*
- *only the last subnode may have a label $n - *$ and all previous subnodes have disjoint integer labels smaller than n.*

Labels of the root node denote absolute coordinates, and can therefore be, instead of integers, the more general Partition Access Specifiers of the corresponding partitioning.
If T is a PAT, let $root(T)$ be the label of the root node.
If N is a node in T let $nlabel(N)$ be the label of this node.

[2] In the usual date format, January is month 1, and the first day in a month is day 1. For dates in this paper, we adopt the convention to count months and days from 0 and not from 1. This way one can interpret month and day numbers as shifts from the first month/day.

A Partition Access Format together with a Partition Access Tree makes up a Tree Partitioning Specification (TPS).

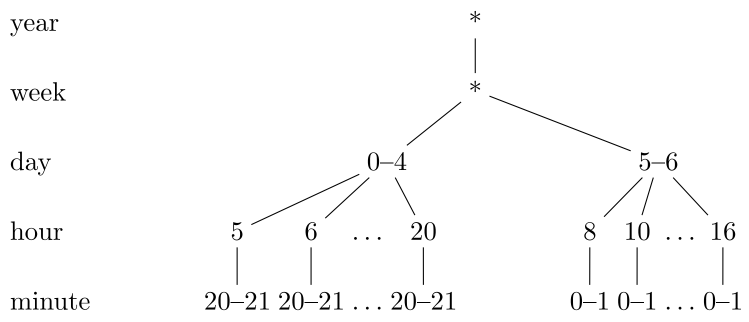
Definition 10 (Tree Partitioning Specification (TPS)). *The specification of a tree partitioning consists of*

– *a Partition Access Format and*
– *one or more Partition Access Trees.*

The PAF must have as least as many elements as the PAT has levels.

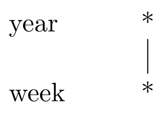
Example 1 (for a Tree Partitioning Specification). A typical PAF is the standard date format year/week,bigger_part_inside/day/hour/minute/second.

The following PAT may define a bus schedule.



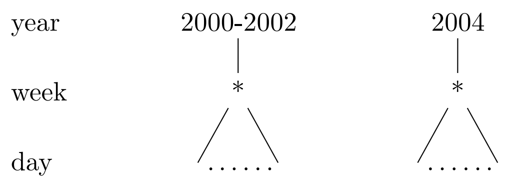
It specifies the following bus schedule: every year, every week, every work day (0–4), there is a bus at 5:20 – 5.21 (2 minutes stay at the bus stop), 6:20 – 6:21 until 20:20 – 20:21, and at the weekends (days 5,6) there is a bus every hour from 8 until 16 hours.

The PAT is actually a finite representation of an infinite forest because the $*$ at the root node indicates the infinite set of integers. Asterisks at the lower levels of the tree indicate only the finitely many coordinates of partitions which are contained in the corresponding partition of the higher level. For example, in the PAT



the '*' at the 'week' level stands for either 0–51 or 0–52, depending on how many weeks the corresponding year has.

A PAT where the root node is labelled with '*' gives in fact rise to an isomorphism with the whole set of integers. In practical implementations we can also allow PAT's where the root node are labelled with a finite set of integers, or where we have a finite forest of such trees:



for example, specifies an isomorphism with a finite segment of the integers, which in turn corresponds to a partitioning of the time axis with infinite partitions at the left and right end.

A Partition Access Tree is usually a finite representation of an infinite tree. All algorithms work on this finite representation. Certain definitions become, however, mathematically simpler if they can refer to the expanded infinite tree.

Definition 11 (Expanded PAT, Admissible Range).
Let T be a Partition Access Tree (or forest) with respect to a Partition Access Format $D \stackrel{\text{def}}{=} P_0, I_0/\ldots/P_n, I_n$. We define Expand$(T)$ by expanding the '' level-wise:*

Level 0*:*
If root$(T) = *$ then replace T by infinitely many copies T_i of T, where root$(T_i) = i$, and i is an integer.
If root$(T) = a - b$ then replace T by copies $T_a, \ldots, T_b$ of T where root$(T_i) = i$.
If root$(T) = n - *$ then replace T by infinitely many copies $T_n, \ldots$ of T where root$(T_i) = i$ and i is an integer.

Level *$l < n$:*
Consider a tree T with a path $k_0/\ldots/k_l$ in T leading to node N. Let T_N be the subtree below and including N.

Let $p_l \stackrel{\text{def}}{=} \Im_{D,l-1}(k_0/\ldots/k_{l-1})$ be the partition specified by $k_0/\ldots/k_{l-1}$ (Def. 6).

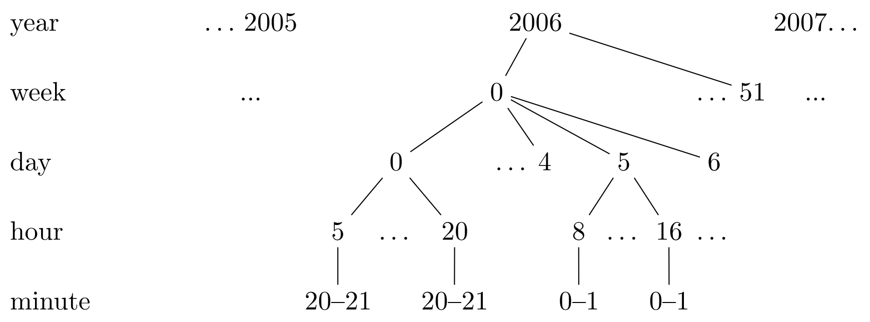
Let $q_0, \ldots, q_n$ be the P_l–partitions in p_l where q_0 is the first P_l–partition in p_l with respect to the inclusion condition I_l and q_n is the last such P_l–partition (q_{n+1} would be the first P_l–partition in the successor of p_l with respect to I_l).

The integers $0, \ldots, n$ are the admissible range *for the node N in Expand(T).*

If $k_l = *$ then replace T_N with copies $T_{N,i}$ of T_N such that root$(T_{N,i}) = i$ and $i = 0, \ldots, n$.
If $k_l = a - b$ then replace T_N with copies $T_{l,i}$ of T_l such that root$(T_{l,i}) = i$ and $i = a, \ldots, min(b, n)$.
If $k_l = a - *$ then replace T_N with copies $T_{N,i}$ of T_N such that root$(T_{N,i}) = i$ and $i = a, \ldots, n$.

We illustrate the expansion operation with the tree of Example 1.

Example 2 (for Expand(T)). If T is the Partition Access Tree of Example 1 then *Expand*(T) looks as follows:



The leaf nodes are not expanded because they determine the length of the partitions. In the bus schedule example, a leaf node with label 20–21 denotes a two minute stop of a bus, whereas two leaf nodes with labels 20 and 21 denote two different buses with a stop of one minute each.

In Def. 6 we have defined the semantics of a Tree Partition Access Specifier with respect to a Partition Access Format. A TPAS has also a meaning with respect to a Tree Partitioning Specification. In this case it denotes either a non-gap partition which corresponds to a leaf node, or a gap partition which corresponds to a gap between two leaf nodes.

Definition 12 (Semantics of a TPAS with respect to a TPS).
Let $S \overset{\text{def}}{=} (D, T)$ *be a Tree Partitioning Specification (Def. 10) where* $D \overset{\text{def}}{=} P_0, I_0 / \dots / P_n, I_n$ *is a Partition Access Format and* T *is a Partition Access Tree.*
Let $d \overset{\text{def}}{=} d_0 / \dots / d_n$ *be a TPAS.*
Let $T' \overset{\text{def}}{=} expand(T)$ *be the expanded PAT (Def.11).*
We define the partition $\Im_S(d)$ *as follows:*

Case 1: $d_0 / \dots / d_{n-1}$ *is a path in* T' *leading to node* N *and* $i \leq d_n \leq j$ *for some subnode* N' *of* N *with* $nlabel(N') =$ *'i–j'. Then* $\Im_S(d) \overset{\text{def}}{=} [a, b[$ *where* a *is the start of* $\Im_D(d_0 / \dots / d_{n-1} / i)$ *and* b *is the end of* $\Im_D(d_0 / \dots / d_{n-1} / j)$.

Case 2: $d_0 / \dots / d_{n-1}$ *is not a path in* T'.

Case 2a: *mismatch at the root node:*
If $T' = \dots T_i, T_j \dots$ *and* $root(T_i) < d_0 < root(T_j)$ *then* $\Im_S(d) \overset{\text{def}}{=} [a, b[$ *where* a *is the end of* $\Im_D(p)$ *with* p *being the rightmost path of* T_i *and* b *is the start of* $\Im_D(q)$ *with* q *being the leftmost path of* T_j[3].

If $T' = \dots T_i$ *and* $root(T_i) < d_0$ *then* $\Im_S(d) \overset{\text{def}}{=} [a, \infty[$ *where* a *is the end of* $\Im_D(p)$ *with* p *being the rightmost path of* T_i.

If $T' = T_i \dots$ *and* $d_0 < root(T_i)$ *then* $\Im_S(d) \overset{\text{def}}{=} [\infty, b[$ *where* b *is the start of* $\Im_D(p)$ *with* p *being the leftmost path of* T_i.

[3] If we speak of left/rightmost path where a leaf node is labelled 'i–j' then 'i' is taken as the end of the leftmost path. 'j' is taken for the end of the rightmost path.

Case 2b: *mismatch at a deeper node:*
If $d_0/\dots/d_k$ *is a path in* T' *leading to node* N*, but* $d_0/\dots/d_{k+1}$ *is not a path in* T'*:*

If the subtrees of N *are* $\dots T_i, T_j \dots$ *and* $root(T_i) < d_0 < root(T_j)$ *then* $\Im_S(d) \overset{\text{def}}{=} [a, b[$ *where* a *is the end of* $\Im_D(p)$ *with* p *being the rightmost path crossing* T_i *and* b *is the start of* $\Im_D(q)$ *with* q *being the leftmost path crossing* T_j*.*

If the subtrees of N *are* $\dots T_i$ *and* $root(T_i) < d_0$ *then* $\Im_S(d) \overset{\text{def}}{=} [a, \infty[$ *where* a *is the end of* $\Im_D(p)$ *with* p *being the rightmost path crossing* T_i*.*

If the subtrees of N *are* $T_i \dots$ *and* $d_0 < root(T_i)$ *then* $\Im_S(d) \overset{\text{def}}{=} [\infty, b[$ *where* b *is the start of* $\Im_D(p)$ *with* p *being the leftmost path crossing* T_i*.*

Now we are ready to define the functions *startOfPartition*(pas) and *endOfPartition*(pas) which map a TPAS to the start/end of the corresponding partition.

Definition 13 **(***startOfPartition* **and** *endOfPartition***).** *Let* S *be a Tree Partitioning Specification (Def. 10). and let* $d \overset{\text{def}}{=} d_0/\dots/d_n$ *be a TPAS with* $\Im_S(d) = [a, b[$ *(Def. 12).*
startOfPartition$_S(d) \overset{\text{def}}{=} a$ *and endOfPartition*$_S(d) \overset{\text{def}}{=} b$*.*

Unfortunately Definition 12 means that there is some redundancy because different TPAS may denote the same partition. In the bus schedule example (Ex. 1) 2006/10/9/5/20 and 2006/10/9/5/21 denote the same partition. In order to be able to check whether two different TPAS specify the same partition, we need to *normalise* the TPAS:

Definition 14 (Normalised TPAS). *Let* $S \overset{\text{def}}{=} (D, T)$ *be a Tree Partitioning Specification (Def. 10) where* $D \overset{\text{def}}{=} P_0, I_0/\dots/P_n, I_n$ *is a Partition Access Format and* T *is a Partition Access Tree. Let* $d \overset{\text{def}}{=} d_0/\dots/d_n$ *be a TPAS.*
Let $T' \overset{\text{def}}{=} expand(T)$ *be the expanded PAT (Def.11).*
We define $Normalise_S(d)$ *as follows:*

Case 1: $d_0/\dots/d_{n-1}$ *is a path in* T' *leading to node* N *and* $i \le d_n \le j$ *for some subnode* N' *of* N *with* $nlabel(N') = i\text{–}j$*. Then* $Normalise_S(d) \overset{\text{def}}{=} d_0/\dots/d_{n-1}/i$*.*

Case 2: $d_0/\dots/d_{n-1}$ *is not a path in* T'*.*

Case 2a: *mismatch at the root node:*
If $T' = \dots T_i, T_j \dots$ *and* $root(T_i) < d_0 < root(T_j)$ *then let* $e_0/\dots/e_n$ *be the rightmost path in* T_i*. Then* $Normalise_S(d) \overset{\text{def}}{=} standardise_D(e_0/\dots/e_{n-1}/(e_n + 1))$*.*

If $T' = \dots T_i$ *and* $root(T_i) < d_0$ *then let* $e_0/\dots/e_n$ *be the rightmost path in* T_i*. Then* $Normalise_S(d) \overset{\text{def}}{=} standardise_D(e_0/\dots/e_{n-1}/(e_n + 1))$*.*

If $T' = T_i \dots$ *and* $d_0 < root(T_i)$ *then let* $e_0/\dots/e_n$ *be the leftmost path in* T_i*. Then* $Normalise_S(d) \overset{\text{def}}{=} standardise_D(e_0/\dots/e_{n-1}/(e_n - 1))$*.*

Case 2b: *mismatch at a deeper node:*
If $d_0/\dots/d_k$ *is a path in* T' *leading to node* N*, but* $d_0/\dots/d_{k+1}$ *is not a path in* T'*:*

If the subtrees of N are $\ldots T_i, T_j \ldots$ and $root(T_i) < d_0 < root(T_j)$ then let $e_0/\ldots/e_n$ be the rightmost path crossing T_i.
Then $Normalise_S(d) \overset{\text{def}}{=} standardise_D(e_0/\ldots/e_{n-1}/(e_n+1))$.

If the subtrees of N are $\ldots T_i$ and $root(T_i) < d_0$ then let $e_0/\ldots/e_n$ be the rightmost path crossing T_i. Then $Normalise_S(d) \overset{\text{def}}{=} standardise_D(e_0/\ldots/e_{n-1}/(e_n+1))$.

The normalised Tree Partition Access Specifiers together with an increment function that goes from a normalised TPAS to the next normalised TPAS, and a corresponding decrement function, is the desired isomorphism to the integers. It can therefore serve as coordinates for the tree partitionings.

Definition 15 (Partition Access Specifier Data Type). *The data type which is isomorphic to the integers, and which therefore can serve as partition access specifier for tree partitionings $S = (D, T)$ consists of*

- *the set of* normalised *Tree Partition Access Specifiers (Def. 14), together with the functions*
- *$increment(pas) \overset{\text{def}}{=} PAS_D(endOfPartition(pas))$ (Def. 7, 13) and*
- *$decrement(pas) \overset{\text{def}}{=} Normalise(PAS_D(startOfPartition(pas) - 1))$.*

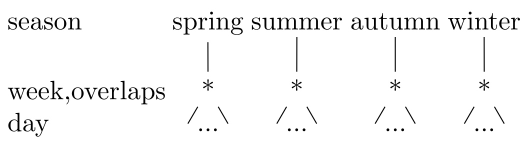
The increment and decrement functions can actually be implemented more efficiently by traversing the partition access tree directly in the same way as in Def. 12 and Def. 14.

4 Generalised Tree Partitionings

The concept which is the kernel of the tree partitioning idea is actually not the partition access tree (Def. 9), but the expanded partition access tree (Def. 11). Except for the leaf nodes, all labels in an expanded PAT are integers. (The labels of the root nodes are at least isomorphic to the integers). The whole idea of tree partitionings works in the same way if it is possible to generate such an expanded PAT from some other PAT like data type. This offers the possibility for a considerable generalisation of tree partitioning specifications. The idea is to use, except for the leaf nodes, instead of concrete numbers as the labels of a node N, a predicate which checks for a given number, whether it falls into the class which is associated with N.

We can use the bus schedule example (Ex. 1) to explain the idea. Suppose there are different time tables for the four seasons. The partitioning which is associated with the top level of the generalised PAT must then model the seasons. To make it more comfortable, we can assume that the partitions of the season partitioning are labelled with the season names 'spring', 'summer', 'autumn', 'winter'. That means, for example, season 0 has label 'spring', season 1 has label 'summer' etc. (A labelling of partitions in this way is possible in the PartLib library.)

The Tree Partitioning Specification for the season dependent bus timetable would then consist of four different PATs.

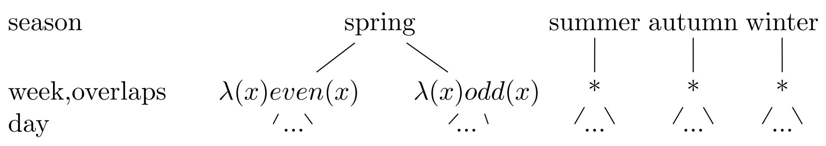


The root labels, 'spring', for example, stands for the predicate which checks for a given season coordinate whether the label of its partition is 'spring' ('summer', 'autumn', 'winter'). These predicates are sufficient to associate a given season coordinate with the corresponding tree.

It does actually not matter whether the boundary between two seasons is in the middle of a week or not. If it is in the middle then the two neighbouring nodes have overlapping parts for some days of the week. Since for each time point t it is clear in which season it is, it is also clear, which of the four trees is relevant. Therefore the overlapping parts of one of the neighbouring trees is irrelevant.

To illustrate the idea of generalised tree partitionings further, assume the spring timetable distinguishes even and odd weeks.

The corresponding tree partitioning specification would then be



For each week number it is possible to identify the corresponding subtree, and this is sufficient for the TPS mechanisms.

In the PartLib library all partitions can be associated with labels. Labels can be strings or any other data structure. The partition labelling mechanism makes also sense for the tree partitionings. In the bus scheduling example, one can, for example, add the bus number as an extra label to the leaf nodes. It is then easy, for example, to compute for a time point t the bus number of the next bus, or to compute the time intervals where a particular bus is at a particular bus station.

The concept of Tree Partitioning Specifiers is currently being integrated into the PartLib library of the CTTN system. Besides the basic implementation of the different specification types it contains several dozens of more or less complex algorithms which work with the application programming interface of the different partitioning data structures.

5 Summary

This paper presents a considerable generalisation to the mechanisms for specifying periodic temporal notions developed so far. The main idea of the generalisation is to represent periodicies no longer by ordinary integers, but by 'partition

access specifiers' (PAS). This is an abstraction for all mathematical structures which are isomorphic to the integers. A particular nontrivial instance of a PAS are the paths in a tree. It is shown how certain forms of trees can be used to specify such complex irregular, but still periodic temporal notions as bus schedules or other time tables.
In this approach we distinguish very clearly

– the partitionings as the mathematical structures behind the periodic temporal notions,
– the specification of these partitionings, and
– the application programming interface for them.

Whereas very different types of specifications have been proposed, and Tree Partitionings Specifications is one of them, it is important to have the same application programming interface for all of them. This offers the possibility to write algorithms for the periodic temporal notions without referring to the details of their specifications.

The presented concepts are currently being integrated into the PartLib library of the CTTN system.

Acknowledgements

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net).

References

1. C. Bettini and R.D.Sibi. Symbolic representation of user-defined time granularities. *Annals of Mathematics and Artificial Intelligence*, 30:53–92, 2000. Kluwer Academic Publishers.
2. Claudio Bettini, Curtis E. Dyreson, William S. Evans, Richard T. Snodgrass, and X. Sean Wang. *Temporal Databases, Rreseach and Practice*, volume 1399 of *LNCS*, chapter A Glossary of Time Granularity Concepts, pages 406–413. Springer Verlag, 1998.
3. Claudio Bettini, Sushil Jajodia, and Sean X. Wang. *Time Granularities in Databases, Data Mining and Temporal Reasoning*. Springer Verlag, 2000.
4. Claudio Bettini, Sergio Mascetti, and X. Sean Wang. Mapping calendar expressions into periodical granularities. In C. Combi and G. Ligozat, editors, *Proc. of the 11th International Symposium on Temporal Representation and Reasoning*, pages 87–95, Los Alamitos, California, 2004. IEEE.
5. Nachum Dershowitz and Edward M. Reingold. *Calendrical Calculations*. Cambridge University Press, 1997.
6. Curtis E. Dyreson, Wikkima S. Evans, Hing Lin, and Richard T. Snodgrass. Efficiently supporting temporal granularities. *IEEE Transactions on Knowledge and Data Engineering*, 12(4):568–587, 2000.

7. Lavinia Egidi and Paolo Terenziani. A lattice of classes of user-defined symbolic periodicities. In C. Combi and G. Ligozat, editors, *Proc. of the 11th International Symposium on Temporal Representation and Reasoning*, pages 13–20, Los Alamitos, California, 2004. IEEE.
8. I.A. Goralwalla, Y. Leontiev, M.T. Ozsu, D. Szafron, and C. Combi. Temporal granularity: Completing the picture. *Journal of Intelligent Information Systems*, 16(1):41–63, 2001.
9. Nick Kline, Jie Li, and Richard Snodgrass. Specifying multiple calendars, calendric systems and field tables and functions in timeadt. Technical Report TR-41, Time Center Report, May 1999.
10. 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.
11. 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.
12. Peng Ning, X. Sean Wang, and Sushil Jajodia. An algebraic representation of calendars. *Annals of Mathematics and Artificial Intelligenc*, 36(1-2):5–38, September 2002. Kluwer Academic Publishers.
13. Hans Jüergen Ohlbach. Computational treatement of temporal notions – the CTTN system. In François Fages, editor, *Proceedings of PPSWR 2005*, Lecture Notes in Computer Science, pages 137–150, 2005. see also URL: http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2005-30.
14. Hans Jürgen Ohlbach. Modelling periodic temporal notions by labelled partitionings – the PartLib library. In S. Artemov, H. Barringer, A. d'Avila Garces, L. C. Lamb, and J. Woods, editors, *Essays in Honour of Dov Gabbay*, volume 2, pages 453–498. College Publications, King's College, London, 2005. ISBN 1-904987-12-5. See also http://www.pms.ifi.lmu.de/publikationen/#PMS-FB-2005-28.
15. Michael D. Soo and Richard T. Snodgrass. Mixed calendar query language support for temporal constants. Technical Report TR 92-07, Dept. of Computer Science, Univ. of Arizona, February 1992.
16. Stephanie Spranger. *Calendars as Types – Data Modeling, Constraint Reasoning, and Type Checking with Calendars.* Dissertation/Ph.D. thesis, Institute of Computer Science, LMU, Munich, Munich, 2005. PhD Thesis, Institute for Informatics, University of Munich, 2005.