A REPRESENTATION FOR COLLECTIONS OF TEMPORAL INTERVALS*

Bruce Leban, David D. McDonald and David R. Forster

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

ABSTRACT

Temporal representation and reasoning are necessary components of systems that consider events that occur in the real world. This work explores ways of considering collections of intervals of time. This line of research is motivated by related work being done by our research group on appointment scheduling and time management. Natural language expressions that refer to collections of intervals are used naturally and routinely in these contexts, and an effective means of representing them is essential.

Previous studies, which considered intervals primarily in isolation, have difficulties in representing some classes of expressions. This occurs not only with expressions that explicitly refer to collections of intervals, such as "the first of every month," but also with expressions that do so only implicitly, such as the U.S. Election Day: "the first Tuesday after the first Monday in November." The traditional solution to this problem has been to provide special means of specifying those forms that are judged to be the most useful (to the exclusion of all other forms).

The "collection representation" builds on previous work in temporal representation by introducing operators that allow the representation of collections of intervals, whether they occur explicitly or implicitly in the expression.

The operators introduced are natural extensions of the relations and operations on intervals. The representation has potential use in scheduling in three areas: graphical display, natural language translation, and reasoning.

I PRIOR WORK

Much of the work on time has focused on temporal reasoning (as opposed to temporal representation). For example, Rescher and Urquhart (1971) and van Benthem (1983) describe temporal logics for reasoning mathematically about time. The logics are based on the concept that instead of a predicate calculus statement being universally true or false, it may be true or false at different moments of time. Temporal quantifiers (much like the universal and existential quantifiers) are used to augment the calculus.

Allen (1983) describes a computational approach to maintaining knowledge about events in time, for use in AI systems that reason about temporal knowledge. Allen's representation takes the concept of a temporal interval as a primitive and explicitly allows representations of indefinite and relative temporal knowledge. A temporal interval is used as the primitive unit because reasoning about points in time frequently yields counter-intuitive or paradoxical results.

Ladkin (1985, 1986a) makes an argument for the use of non-convex intervals for reasoning. A convex interval is an interval in the usual sense: a contiguous period of time. A non-convex interval is an arbitrary union of convex intervals.

In this paper, it is assumed that a temporal structure based on convex intervals has been defined that has a useful set of operations and relations (see appendix). We believe that the work could be extended to temporal structures based on time-points or non-convex intervals.

II COLLECTIONS OF INTERVALS

An interval t is denoted by $\langle t_\alpha, t_\beta \rangle$ or $\langle t_\alpha; t_\delta \rangle$ where t_α, t_β and $t_\alpha + t_\delta$ are real numbers denoting moments in time; the interval starts at time t_α and extends through time t_β or $t_\alpha + t_\delta$.**

A *collection* of intervals is a structured set of intervals. The *order* of a collection is a measure of the depth of the structure. An order 1 collection is an ordered list of intervals. This is somewhat similar to a non-convex interval except that the maximal convex subintervals of

* This work was supported in part by the Air Force Systems Control, Rome Air Force Development Center, Griffiss AFB, New York, 13441 and the Air Force Office of Scientific Research, Bolling AFB, DC 20332 under Contract No. F30602-85-C-0008 and by the National Science Foundation under Support and Maintenance Grant DCR-8318776.

** We ignore the sticky questions of whether the intervals are open or closed and whether time is represented in a continuous or discrete fashion, as these issues are largely irrelevant to the work discussed here. We assume that if t and u are intervals and $t_\beta = u_\alpha$ then $t \cup u = \langle t_\alpha, u_\beta \rangle$.

a non-convex interval are disjoint and the order they are given in is immaterial. An order n collection ($n > 1$) is an ordered list of order $n-1$ collections. The notation used for collections is essentially set notation, except for the understanding that the order of elements is maintained. For example,

$$\{\{x_1, x_2\}, \{x_3, x_4\}, \{x_5\}\}$$

is an order 2 collection. The collection of Thursdays (which contains all the Thursdays in order) is an example of an order 1 collection. The collection of months where each month is represented by a collection of the days in that month (in order) is an order 2 collection.

A. A Formula Approach

Many useful collections can be described by arithmetical formulae, but there are subtle difficulties with this. We reject this approach for the reasons outlined in this section.

Given an appropriate definition for *day* representing the length of one day and, for convenience, assume that time t_0 is Saturday, December 31, 1904, midnight, the collection of Thursdays can be described by the formula:

$$Thursdays = \{\langle \alpha; 1day \rangle \mid \alpha = 5days + t_0 \pmod{7days}\}$$

We can generalize *Thursdays* by replacing the 5 with any other value. In other words, it can be understood that *Tuesdays* is an essentially similar collection to *Thursdays*.

The same approach applied to construct the collection of all Januarys is less successful. Since every fourth year is a different length, one possible formula is:

$$Januarys = \{\langle \alpha; 31days \rangle \mid (\alpha + t_0 \bmod 1461days) \in \{0, 365, 730, 1095\}\}$$

This formula is considerably more complicated than the one given for *Thursdays*.* More importantly, it fails to provide a means of conveniently recognizing *Augusts* as a generalization of *Januarys*. To generalize from *Januarys* we would need to replace each of the values (except 1461) with appropriate new values: the chance of an arbitrary substitution producing a reasonable generalization is quite small. Essentially, the formula is in a "compiled" form that is quite distant from how the concept would naturally be expressed.

The formulae become even more complicated when new collections must be built from existing collections. For example, consider "the first Thursday of every January." This requires combining the collection of Thursdays and the collection of Januarys to produce a new collection. Furthermore, the system must allow for collections to be combined in fairly arbitrary ways, since it will not be possible to predict all useful specifications.

* It would be even more complicated if it were correct: the Gregorian calendar specifies that only 97 out of every 400 years are leap years.

III THE COLLECTION REPRESENTATION

The foundation of the collection representation is a set of primitive collections called *calendars*. A calendar is a collection consisting of an infinite sequence of intervals that span the timeline, i.e., t_i meets t_{i+1} for any two consecutive intervals. A calendar may have a first interval (the first moment in time the system is prepared to consider), but does not have a last interval. *Days*, *Months* and *Chinese-Calendar-Years* are instances of calendars.

Two new classes of operators, *slicing* and *dicing*, are defined to operate on collections of intervals. The dicing operators provide means of generating collections from intervals, for example, to break a collection of intervals into smaller intervals. In Figure 1, a dicing operation is illustrated between the first two steps. This operation replaces each interval on the left (a week) with a collection of subintervals (the days in that week).

The slicing operators provide means of selecting intervals from collections of intervals, for example, to select the first interval of a collection. In Figure 1, a slicing operation is illustrated between the second two steps. This operation replaces each order 1 collection (a collection of the days in each week) with a single interval (the fifth day of each week).

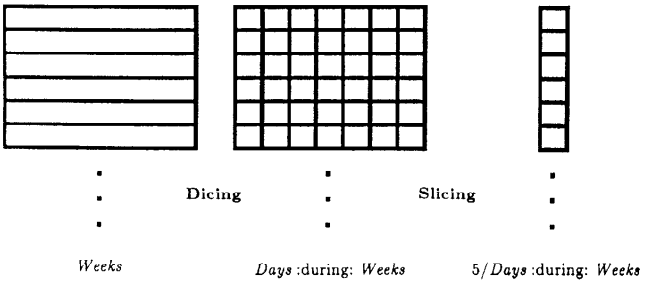


Figure 1. Slicing and Dicing

The terms "slicing" and "dicing" are chosen for both their euphonic and metaphoric appeal.** The operators have a right-to-left precedence. Each operator corresponds roughly to a preposition, so these expressions can be read naturally by someone who speaks a prepositional language (e.g., English).

New collections can be built by combining other collections using these operators. The calendars serve as a basis for this construction. Since the calendars are not sufficient for reasoning about statements that reference collections that might not yet have been defined or might include unknown intervals in the future (e.g., "when Diana is at work"), collections can also be built by predicate reference.

** If these terms seem to have conflicting meanings, "Slicing" can be thought of as corresponding to "Selection" and "Dicing" to "Dividing up".

A. Primitive Collections

A calendar is defined by specifying the intervals of which it is composed. The notation $\langle\langle \alpha; \delta_1; \delta_2; \ldots; \delta_n \rangle\rangle$ denotes the calendar

$$\{\langle \alpha; \delta_1 \rangle, \langle \alpha+\delta_1; \delta_2 \rangle, \ldots, \langle \alpha+\sum_{i\leq n-1} \delta_i, \delta_n \rangle, \langle \alpha+\sum_{i\leq n} \delta_i, \delta_1 \rangle, \ldots\}.$$

The list of δ-values is treated as if it were a circular list.

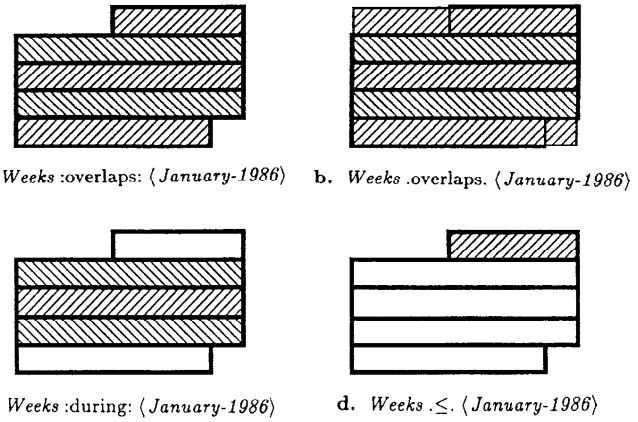
A calendar can also be defined by specifying how it is to be constructed from another calendar. This is denoted by $\langle\langle C; s_1; s_2; \ldots; s_n \rangle\rangle$ to indicate that the first interval of this calendar is the union of the first s_1 intervals of C; the second interval is the union of the next s_2 intervals of C, etc. As above, the list of s-values is treated as a circular list.

If we assume that the unit of measure is 1 second, we might have the following definitions:

$$\begin{aligned} \textit{Days} &\equiv \langle\langle t_0; 86400 \rangle\rangle \\ \textit{Months} &\equiv \langle\langle \textit{Days}; 31;28;31;30;31;30;31;31;30;31;30;31; \\ &\qquad 31;28;31;30;31;30;31;31;30;31;30;31; \\ &\qquad 31;28;31;30;31;30;31;31;30;31;30;31; \\ &\qquad 31;29;31;30;31;30;31;31;30;31;30;31 \rangle\rangle \end{aligned}$$

These definitions are intensional rather than extensional. That is, while a calendar defines an infinite data structure, it does not require that an implementation actually build the complete structure, but only that it build those portions of the structure it needs.

Collections can also be constructed from a predicate. The collection $\langle\langle \textit{Condition} \rangle\rangle$ is the minimal collection of intervals C that satisfies the property that there does not exist an interval t disjoint from C, such that *Condition* is true during t. This definition is carefully constructed to avoid the question of whether the predicate operates on intervals or points.



a. *Weeks* :overlaps: ⟨*January-1986*⟩ b. *Weeks* .overlaps. ⟨*January-1986*⟩

c. *Weeks* :during: ⟨*January-1986*⟩ d. *Weeks* .≤. ⟨*January-1986*⟩

Figure 2. Dicing Operators

B. The Dicing Operations

The dicing operators are extensions of the relations on intervals (listed in the appendix). A dicing operator takes an order 1 collection as its left argument, an interval as its right argument and produces an order 1 collection as a result. A dicing operator can also take a collection as the right argument, in which case it operates on each interval in that collection.

For each relational operator (R) there are two dicing operators: strict (:R:) and relaxed (.R.). If C is an order 1 collection and t is an interval, the dicing operators are defined by:

$$C :\mathrm{R}: t \equiv \{c \cap t \mid c \in C \wedge c \,\mathrm{R}\, t\} \setminus \{\varepsilon\}$$
$$C .\mathrm{R}. t \equiv \{c \mid c \in C \wedge c \,\mathrm{R}\, t\} \setminus \{\varepsilon\}$$

The effect of a strict dicing operator is to break up t into pieces according to C. An illustrative example occurs when C is a calendar. The expression *Weeks* :overlaps: ⟨*January-1986*⟩ will break up the month on the boundaries of the weeks, i.e., it will give those weeks or parts of weeks that overlap the month. (See Figure 2a.) The effect of a relaxed dicing operator is to select intervals from C that have the appropriate relation with t. Thus *Weeks* .overlaps. ⟨*January-1986*⟩ will break up the month in the same way as above, but for the weeks at the beginning and end of the month it will give the entire week (including that part not overlapping the month). (See Figure 2b.) In contrast, *Weeks* :during: ⟨*January-1986*⟩ will give only the weeks that are completely contained in the month. (See Figure 2c.)' Finally, *Weeks* :≤: ⟨*January-1986*⟩ will give only the partial week at the beginning of the month. (See Figure 2d.)

C. The Slicing Operations

The slicing operators, denoted f/C and $[f]/C$, operate on any collection, replacing each of the contained order 1 collections with the result of the application of the slicing operator. Operating on an order 1 collection yields either a single interval or an order 1 collection (usually a subcollection of the original order 1 collection). The expression f/C applies the selection function f to the collection and returns a single interval, while $[f]/C$ returns a collection. F may be a predicate, in which case it constructs a collection containing the intervals which satisfy the predicate. The expression $[f_1, f_2, \ldots, f_n]/C$ is the collection consisting of the individual applications of f_1, f_2, ..., f_n to C in order.

In some cases, a selection function may not have a result (e.g., the 29ths of Februarys), in which case the result is defined to be the empty interval ε. Note that since the dicing operators will never produce a collection that contains ε, any result that includes ε is a sign of a failed selection operation.

The integers are defined as selection functions so that n/C selects the nth interval in C and $-n/C$ selects the nth interval from the end. The function *the* is defined so that *the*/C selects the single interval of C, and produces ϵ if C contains other than a single interval.

The function *any* is used to select intervals nondeterministically. *any*/C selects a single interval of C. [*any* n]/C selects n intervals of C. [*any* $-n$]/C selects all but n intervals of C. The *any* slicing operator has a subtly different operation when used in a declarative statement — in that case, it refers to an interval without specifying which one. This usage of *any* has a close relationship to the existential quantifier of the predicate calculus.

D. Examples of Collections

Table 1 gives a list of English phrases and their corresponding expressions in the collection representation.

IV APPLICATIONS

The reason for constructing this representation is to provide a framework for a scheduling system. The previous sections have shown how terms commonly used in scheduling can be easily expressed. The representation was designed to address three areas of our group's research on scheduling: graphical display, natural language translation (primarily generation), and reasoning (about schedules).

The illustrations in Figures 1 and 2 indicate the type of graphical display that would be generated by the system. The definition of each calendar can be made to contain simple graphical display information, such as the shape and orientation of any "boxes" in which they and their contents are shown. The boxes in Figures 1 and 2 are unlabelled. An interval of a calendar could also carry tags that could be used to label the boxes or to organize the data in a tabular form.

The bus schedule of Figure 3 provides a good illustration of this. The schedule is constructed as an order 2 collection, where each interval has been tagged. The collection prefers to display intervals with the same tag in the same column. The intervals in turn prefer to display only their start times. Notice that in several places a table entry is blank. Despite this, displaying the table presents no problem.

Hampshire	Amherst	UMass	Smith	Mt. Holyoke
—	—	8:20	8:35	8:45
10:00	10:10	10:20	10:35	10:45
11:00	11:10	11:20	11:35	11:45
... Every hour ...				
6:00	6:10	6:20	6:35	6:45
7:00	7:10	7:20	—	7:45
8:00	8:10	8:20	8:35	—
11:00	11:10	11:20	11:35	11:45

Figure 3. A Bus Schedule

The appointment calendar display of Figure 4 would be treated in a similar fashion. In this case, the collection of appointments is superimposed on another collection to provide the time grid, with the roles of tags and starting times reversed in the displayed table.

The English text in Table 1 indicates the type of natural language that could be produced or processed by the system. Expressions in the collection representation can be almost literally translated into natural language with comprehensible results. Similarly, statements can be eas-

Table 1. **English**	**Collection Representation**
Mondays	2/*Days* :during: *Weeks*
Januarys	1/*Months* :during: *Years*
First Monday in January 1986	1/*Mondays* :during: *Januarys* :during: 1986/*Years*
or equivalently:	1/(2/*Days* :during: *Weeks*) :during: 1/*Months* :during: 1986/*Years*
First of every month	1/*Days* :during: *Months*
First Monday of every month	1/*Mondays* :during: *Months*
Last two Mondays of every month	[−1, −2]/*Mondays* :during: *Months*
Week of the 15th of each month	*the*/*Weeks* .overlaps. 15/*Days* :during: *Months*
First full week of each month	1/*Weeks* :during: *Months*
Week of the first of the month	1/*Weeks* .overlaps. 1/*Days* :during: *Months*
First week of the month	1/*Weeks* .overlaps. *Months*
U.S. Election Day	1/*Tuesdays* .>. 1/*Mondays* :during: *November*
The first (or only) day of t	1/*Days* .overlaps. t
The day after t	1/*Days* .>meets. −1/*Days* .overlaps. t
Any day of the week	*any*/*Days* :during: *Weeks*
Any day this week	*any*/*Days* :during: *Weeks* .overlaps. ⟨*Today*⟩

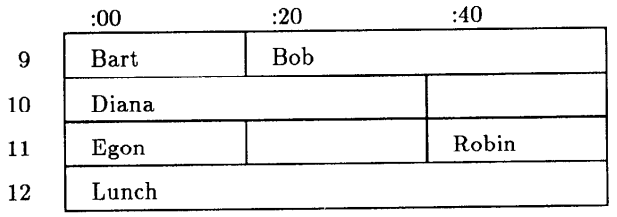


Figure 4. An Appointment Calendar

ily translated since the temporal components of the statement are not distributed across a number of quantifiers and predicates. For example, the statement

$\langle\langle$ *Roy-worked* $\rangle\rangle$ contains *Weekend-Days* :during: $\langle$ *January* $\rangle$

can be glossed as "The time that Roy worked included the weekend days in January."

Since the expressions are stored symbolically, the system need only generate the actual intervals that it needs. For example, for the expression

23/*Seconds* :during: 4570/*Minutes* :during: 1986/*Years*

the system naturally would not generate a data structure containing the 31536000 seconds in 1986 before selecting the one desired. If the system was asked whether two expressions conflicted and could not determine this by purely symbolic means, it still would not need to generate all the intervals in each collection. Only those subcollections and intervals that have been determined to be possible candidates for conflicts need to be generated (and this process can be done recursively).

If scheduling conflicts occur, the system can replace specific slicing operators with the *any* operator. For example, the system could make the following successive generalizations in searching for a non-conflicting schedule:

the/*Mondays* :during: 1/*Weeks* :during: *Months*
the/*Anyday* :during: 1/*Weeks* :during: *Months*
the/*Mondays* :during: *any*/*Weeks* :during: *Months*
the/*Anyday* :during: *any*/*Weeks* :during: *Months*

Our motivation for this work has been to provide a framework for the scheduling system. We are in the process of building a scheduling system around the representation. We believe that the consideration of collections of intervals is essential to the scheduling domain and that the notation and accompanying semantics introduced in this paper provide a natural medium for that consideration.

ACKNOWLEDGMENTS

We would like to thank Scott Anderson, Carol Broverman, John Brolio, David Lewis, James Pustejovsky, Penelope Sibun, Philip Werner, Mary-Anne Wolf, and Bev Woolf for their assistance in this research and/or the preparation of this paper. We would also like to thank Peter Ladkin for sending us advance copies of his papers presented at this conference.

APPENDIX

The intersection of two intervals is defined by:

$$t \cap u \equiv \langle \max(t_\alpha, u_\alpha), \min(t_\beta, u_\beta) \rangle$$

The cover of two intervals is defined by:

$$t \uplus u \equiv \langle \min(t_\alpha, u_\alpha), \max(t_\beta, u_\beta) \rangle$$

The union of two intervals $(t \cup u)$ is defined only if the intervals overlap or meet and is equal to the cover of the two intervals. The empty interval $\varepsilon = \langle \infty, -\infty \rangle$ and any interval that has $\alpha \geq \beta$ is automatically replaced by ε. This definition is motivated by the desire to have $t \cap \varepsilon = \varepsilon$ and $t \uplus \varepsilon = t$, for any t.

We use the following binary relations on intervals:

$$\begin{aligned} t \text{ overlaps } u &\equiv t \cap u \neq \varepsilon \\ t \text{ during } u &\equiv (t_\alpha \geq u_\alpha) \wedge (t_\beta \leq u_\beta) \\ t \text{ contains } u &\equiv u \text{ during } t \\ t < u &\equiv t_\beta \leq u_\alpha \\ t > u &\equiv t_\alpha \geq u_\beta \\ t \leq u &\equiv (t_\alpha \leq u_\alpha) \wedge (t_\beta \leq u_\beta) \\ t \geq u &\equiv (t_\alpha \geq u_\alpha) \wedge (t_\beta \geq u_\beta) \\ t \text{ meets } u &\equiv (t_\beta = u_\alpha) \end{aligned}$$

The during, $\leq$ and $\geq$ operations form partial orders. Note that $t \leq u$ is *not* equivalent to $(t < u) \vee (t = u)$; however, $t < u$ is equivalent to $(t \leq u) \wedge \neg(t \text{ overlaps } u)$.

REFERENCES

Allen, James F., 1985, "Maintaining Knowledge about Temporal Intervals" in Brachman and Levesque, *Readings in Knowledge Representation.* Morgan Kaufmann, pp. 509–521.

van Benthem, J.F.A.K., 1983, *The Logic of Time.* D. Reidel, Boston.

Ladkin, Peter, 1985, "Comments on the Representation of Time" in Proceedings of the 1985 Distributed Artificial Intelligence Workshop, Sea Ranch, California, pp. 137–156.

Ladkin, Peter, 1986a, "Primitives and Units for Time Specification" in the Proceedings of the National Conference on Artificial Intelligence, Philadelphia, Pennsylvania.

Ladkin, Peter, 1986b, "Time Representation: A Taxonomy of Interval Relations" in the Proceedings of the National Conference on Artificial Intelligence, Philadelphia, Pennsylvania.

Rescher, Nicholas, and Urquhart, Alasdair, 1971, *Temporal Logic.* Springer-Verlag, New York.