SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 20(9), 899–928 (SEPTEMBER 1990)

Calendrical Calculations

NACHUM DERSHOWITZ AND EDWARD M. REINGOLD
Department of Computer Science, University of Illinois at Urbana-Champaign,
1304 W. Springfield Avenue, Urbana, IL 61801-2987, U.S.A.

SUMMARY

A unified, algorithmic presentation is given for the Gregorian (current civil), ISO, Julian (old civil), Islamic (Moslem), and Hebrew (Jewish) calendars. Easy conversion among these calendars is a byproduct of the approach, as is the determination of secular and religious holidays.

KEY WORDS Calendar Holidays Gregorian calendar Hebrew calendar Islamic calendar ISO calendar Julian calendar

Teach us to number our days, that we may attain a wise heart.
—Psalms 90:12

INTRODUCTION

Calendrical calculations are ubiquitous. Banks need to calculate interest on a daily basis. Operating systems need to switch to and from daylight savings time. Dates of secular and religious holidays need to be computed for consideration in planning events. Paychecks need to be issued on weekly, biweekly, or monthly schedules. Bills and statements must be generated periodically. Most of these calculations are not difficult because the rules of our civil calendar (the Gregorian calendar) are straightforward.

Complications begin when we need to know the day of the week on which a given date falls or when various religious holidays based on other calendars occur. These complications lead to difficult programming tasks—not difficult in an algorithmic sense, but difficult because it can be extremely tedious to delve, for example, into the complexities of the Hebrew calendar and its relation to the civil calendar.

The purpose of this paper is to present, in a completely algorithmic form, a description of five basic calendars and how they relate to one another: the present civil calendar (Gregorian), the recent ISO commercial calendar, the old civil calendar (Julian), the Islamic (Moslem) calendar, and the Hebrew (Jewish) calendar. Information that is sufficiently detailed to allow computer implementation is difficult to find for the Islamic and Hebrew calendars since the published material is often inaccessible, ecclesiastically oriented, incomplete, inaccurate, based on extensive tables, overburdened with extraneous material, focused on shortcuts for hand calculation to

0038–0644/90/090899–30$15.00

Received 31 August 1989
Revised 27 March 1990

avoid complicated arithmetic or to check results, or difficult to find in English. Most existing computer programs are proprietary, incomplete, or inaccurate.

The need for such a secular presentation in the public domain was made clear to us when the second author, in implementing a calendar/diary feature for GNU Emacs,[1] found difficulty in gathering and interpreting appropriate source materials that describe the interrelationships among the various calendars and the determination of the dates of holidays. The material presented in this paper, in the form of COMMON LISP[2] functions, unifies the calculations for all five calendars.* We have chosen Lisp as the vehicle for implementation because it encourages functional programming and has a trivial syntax, nearly self-evident semantics, historical durability, and wide distribution.

It is not the intention of this paper to give a detailed historical treatment of the material, nor, for that matter, a mathematical one; our goal is to give a computational treatment that will prove useful to programmers. Thus, although we give some necessary historical, religious, mathematical, and astronomical details in the text, the focus of the presentation is in the Lisp functions. Full historical/religious details and mathematical/astronomical underpinnings of the calendars can be pursued in the references. We have chosen not to optimize the code at the expense of algorithmic clarity; consequently, considerable improvements in economy are possible (some possibilities are pointed out).

In the next section we describe the underlying unifying idea of all the calculations. The details of specific calendars are presented in subsequent sections. Historically, the oldest of the calendars that we consider is the Julian (the roots of which date back to the ancient Roman empire). Next oldest is the Hebrew calendar (fourth century), followed by the Islamic calendar (seventh century), followed by the Gregorian modification to the Julian calendar (sixteenth century). Finally, the International Organization for Standardization's ISO calendar is of twentieth century origin. For expository purposes, however, we present the Gregorian calendar first because it is the most popular calendar currently in use; then we give the ISO calendar which depends wholly on the Gregorian. Since the Julian calendar is so close in substance to the Gregorian, we present it next. Then we give the Islamic calendar which, because of its simplicity, is easy to implement. Finally, we present the Hebrew calendar, the most complicated of the five calendars, and the most difficult to implement. In the penultimate section, we give algorithms for calculating the dates of all major and many minor secular and religious holidays. The final section contains descriptions of two other calendars for which we do not provide algorithmic details.

ABSOLUTE DAY NUMBERS

Over the centuries, human beings have devised an enormous variety of methods for specifying dates.[3–6] (An exceptional survey can be found in the *Encyclopædia of Religion and Ethics*,[7] vol. III, pp. 61–141 and vol. V, pp. 835–894.) None are ideal

---

*To insure correctness, all code in this paper was typeset directly from working Lisp functions. We will gladly provide these Lisp functions in electronic form: send an empty electronic mail message to `reingold@cs.uiuc.edu` with the subject line containing precisely the phrase 'send-cal'; your message will be answered automatically.

computationally, however, because all have idiosyncrasies resulting from attempts to coordinate a convenient human labeling with lunar and/or solar phenomena. All of the calendars that we consider have an integral number of days in a month and an integral number of months in a year, but the astronomical events with which they are supposed to correlate do not follow such a convenient pattern, nor are the precise lengths of astronomical cycles constant over time. Rather, the *mean* length of a (synodic) month is currently 29.5306 (mean) days and the current *mean* length of a (tropical) year is 365.2422 days. The different calendars differ in the accuracy with which their months and years approximate these figures.

For a computer implementation, the easiest way to reckon time is simply to count days: Establish an arbitrary starting point as day 1 and specify a date by giving a day number relative to that starting point;[8] a single thirty-two bit integer allows the representation of more than 11.7 million years. Such a reckoning of time is, evidently, extremely awkward for human beings and is not in common use, except among astronomers who use *Julian day numbers* to specify dates.[9]

We have chosen Monday, January 1, 1 C.E.* (Gregorian) as our absolute date 1 and count forward day-by-day from there. Of course, this is anachronistic because there was no year 1 C.E. on the Gregorian calendar—the Gregorian calendar was devised only in the sixteenth century—so by January 1, 1 C.E. (Gregorian) we mean the day we get if we extrapolate backwards from the present; this day turns out to be Monday, January 3, 1 C.E. (Julian).

We should thus think of the passage of time as a sequence of days numbered 1, 2, 3, ... that the various human-oriented calendars label differently. For example, absolute day 710347 is called November 12, 1945 C.E. on the Gregorian calendar, day 1 of week 46 of 1945 C.E. on the ISO calendar, October 30, 1945 C.E. on the Julian calendar, Dhu al-Hijjah 6, 1364 A.H.† on the Islamic calendar, and Kislev 7, 5706 A.M.‡ on the Hebrew calendar. All that is required for calendrical conversion is to be able to convert to and from this absolute calendar. We give, in subsequent sections, Lisp functions to do the conversions for the Gregorian, ISO, Julian, Islamic, and Hebrew calendars. The algorithms given in this paper do *not* generally work for non-positive absolute dates.

The date Monday, January 1, 1 C.E. (Gregorian), though arbitrarily chosen as our starting point, has two desirable characteristics. First, it is early enough that almost all dates of interest are represented by positive integers; in any case, using any earlier date would be problematic because of historical irregularities in the application of the Julian leap year rule. Second, since the day is a Monday, determining the day of the week amounts to taking the absolute date modulo seven—zero is Sunday, one is Monday, and so forth.

LISP PRELIMINARIES

For readers unfamiliar with Lisp, this section provides the bare necessities. Other details will be mentioned in passing as they are used; a complete description can be found in COMMON LISP: *The Language.*[2]

---

*Common era; or, A.D.

†*Anno hegirae*; in the year of the Hegira (Mohammed's flight to Medina).

‡*Anno mundi*; in the (traditional) year of the world (since creation).

All functions in Lisp are written in prefix notation: If `f` is a defined function, then

```
(f e0 e1 e2 ... en)
```

applies `f` to the $n+1$ arguments `e0`, `e1`, `e2`, ..., `en`. Thus

```
(+ 1 -2 3)
```

adds the three numbers and returns the value 2;

```
(<= 1 2 3)
```

checks that the three numbers are in nondecreasing order and yields true (`t` in Lisp).

Lists are Lisp's main data structure. To construct a list `(e0 e1 e2 ... en)` the expression

```
(list e0 e1 e2 ... en)
```

is used. The function `(nth i l)` extracts the ith element of the list `l`, indexing from zero; the predicate `(member x l)` tests if `x` is an element of `l`. To get the first (indexed zero), second, or third elements of a list, we use the functions `first`, `second`, and `third`, respectively. The empty list is represented by `nil`.

Functions are defined using the `defun` command, which has the following syntax:

```
(defun function-name (param1 ... paramn)
   expression)
```

For example, we can define a function (unavailable in COMMON LISP) to return the (truncated) integer quotient of two integers:

```
(defun quotient (m n)
   (floor m n))
```

We will represent all dates on the Gregorian, Julian, Islamic, and Hebrew calendars by a list of the form (*month day year*) in which *month*, *day*, and *year* are each integers. (COMMON LISP places no *a priori* upper bound on the size of integers; none of our calculations require more than 32-bit integers for dates in the next twenty thousand years; 24 bits suffice for all of the calculations, except as noted.) To extract the individual components of such a date we use the following access functions:

```
(defun extract-month (date)
;; Month field of date = (month day year).
  (first date))
(defun extract-day (date)
;; Day field of date = (month day year).
  (second date))
(defun extract-year (date)
;; Year field of date = (month day year).
  (third date))
```

Notice that the double semicolons demarcate comments.

For convenience in expressing our calendar functions in Lisp, we introduce a macro to compute sums. The expression

```
(sum f i k p)
```

computes $\sum_{i \geq k,\, p(i)} f(i)$; that is, the expression $f(i)$ is summed for all $i = k$, $k+1$, ..., continuing only as long as the condition $p(i)$ holds. The (opaque) COMMON LISP definition of `sum` is as follows:

```
(defmacro sum (expression index initial condition)
;; Sum expression for index = initial and successive integers,
;; as long as condition holds.
  (let* ((temp (gensym)))
    `(do ((,temp 0 (+ ,temp ,expression))
          (,index ,initial (1+ ,index)))
         ((not ,condition) ,temp))))
```

THE GREGORIAN CALENDAR

The calendar in use today in most countries is the new style, or Gregorian, calendar designed by a commission assembled by Pope Gregory XIII in the sixteenth century.[3, 5, 6, 10–13] This strictly solar calendar is based on a 365-day common year divided into twelve months of lengths 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, and 31 days, and on 366 days in leap years, the extra day being added to make the second month 29 days long:

(1) January	31 days	(7) July	31 days
(2) February	28 {29} days	(8) August	31 days
(3) March	31 days	(9) September	30 days
(4) April	30 days	(10) October	31 days
(5) May	31 days	(11) November	30 days
(6) June	30 days	(12) December	31 days

The leap-year structure is given in curly brackets—a year is a leap year if it is divisible by 4 and is not a century year (multiple of 100) or if it is divisible by 400. For example, 1900 was not a leap year, while 2000 will be. The Gregorian calendar differs from its predecessor, the old style or Julian calendar, only in that the Julian calendar did not include the century rule for leap years—all century years were leap years.

The Julian calendar was instituted in 45 B.C.E.* by Julius Cæsar on January 1, 709 A.U.C.†; it was a modification of an ancient Egyptian calendar. Since every fourth year was a leap year, a cycle of 4 years contained $4 \times 365 + 1 = 1461$ days, giving an average length of year of 365.25 days. This is somewhat more than the mean

*Before the common era; or, B.C.

†*Ab urbe condita*; from the (traditional) founding of the city (of Rome).

length of the solar year, and over the centuries the calendar slipped with respect to the solar year. By the sixteenth century, the date of the vernal (spring) equinox had shifted from around March 21 to around March 11. If this error were not corrected, eventually Easter, the date of which depends on the vernal equinox, would migrate through the whole calendar year. Pope Gregory instituted only a minor change in the calendar—century years not divisible by 400 would no longer be leap years. Thus, three out of four century years are common years, giving a cycle of 400 years containing $400 \times 365 + 97 = 146097$ days and an average year length of $146097/400 = 365.2425$ days. He also corrected the accumulated 10-day error in the calendar by proclaiming that Thursday, October 4, 1582 C.E., the last date in the old style (Julian calendar), would be followed by Friday, October 15, 1582 C.E., the first day of the new style (Gregorian) calendar. Catholic countries followed his rule, but Protestant countries resisted: Spain, Portugal, and Italy adopted it immediately, as did the Catholic states in Germany. The Protestant parts of Germany waited until 1700 to adopt it, Great Britain and its colonies (including the United States) waited until 1752, Russia held out until after the revolution in 1918, and Bulgaria until 1920 (an extensive list of dates of adoption of the Gregorian calendar can be found in the *Explanatory Supplement to the Astronomical Ephemeris and the American Ephemeris and Nautical Almanac*[9]).

To convert from a Gregorian date to an absolute date, we first need a function that gives the last day (that is, the number of days) for any Gregorian month. This is easily written:

```
(defun last-day-of-gregorian-month (month year)
;; Last day in Gregorian month during year.
  (if ;; February in a leap year
      (and (= month 2)
           (= (mod year 4) 0)
           (not (member (mod year 400) (list 100 200 300))))
;; Then return
      29
;; Else return
    (nth (1- month)
         (list 31 28 31 30 31 30 31 31 30 31 30 31))))
```

The function `if` has three arguments, a Boolean condition, a then-expression, and an else-expression. The function `1-` decrements an integer by one (the similar function `1+` increments by one). Here, and elsewhere, arrays would be more efficient than lists.

The calculation of the absolute date from the Gregorian date (which has been described[8] as 'impractical') can now be done by counting the number of days in prior years—both common and leap years, the number of days in prior months of the current year, and the number of days in the current month:

```
(defun absolute-from-gregorian (date)
;; Absolute date equivalent to the Gregorian date.
  (let* ((month (extract-month date))
         (year (extract-year date)))
;;  Return
    (+ (extract-day date)    ;; Days so far this month.
       (sum                  ;; Days in prior months this year.
```

```
 (last-day-of-gregorian-month m year) m 1 (< m month))
(* 365 (1- year))     ;; Days in prior years.
(quotient (1- year) 4);; Julian leap days in prior years...
(-                    ;; ...minus prior century years...
 (quotient (1- year) 100))
(quotient             ;; ...plus prior years divisible...
 (1- year) 400))))    ;; ...by 400.
```

The Common Lisp construct `let*` defines a sequence of constants (possibly in terms of previously defined constants) and ends with an expression the value of which is returned by the construct.

Calculating the Gregorian date from the absolute date d involves sequentially determining the year, month, and day of the month. The year is first closely approximated from below by $\lfloor d/366 \rfloor$ and then found precisely by stepping through subsequent years (that is, by a linear search). The month is then found by a similar linear process, and the day of the month is determined by subtraction:

```
(defun gregorian-from-absolute (date)
;; Gregorian (month day year) corresponding absolute date.
  (let* ((approx (quotient date 366));; Approximation from below.
         (year         ;; Search forward from the approximation.
          (+ approx
             (sum 1 y approx
                  (>= date
                      (absolute-from-gregorian
                       (list 1 1 (1+ y)))))))
         (month        ;; Search forward from January.
          (1+ (sum 1 m 1
                   (> date
                      (absolute-from-gregorian
                       (list m
                             (last-day-of-gregorian-month m year)
                             year))))))
         (day          ;; Calculate the day by subtraction.
          (- date (1- (absolute-from-gregorian
                       (list month 1 year))))))
;;  Return
    (list month day year)))
```

It is not hard to determine better approximations for the year, and the same is true for the approximations we will use for other calendars. In fact, the year can be determined *exactly* without searching, but at considerable expense in the clarity of the code.*

*The exact determination of the Gregorian year from the absolute *date* is an exercise in base conversion in a mixed-radix system[14]:

$$
\begin{aligned}
n_{400} &= \lfloor (date - 1)/146097 \rfloor && \{\text{number of completed 400 year cycles}\} \\
d_1 &= (date - 1) \bmod 146097 && \{\text{days not included in } n_{400}\}
\end{aligned}
$$

THE ISO CALENDAR

The International Organization for Standardization (ISO) calendar, popular in Sweden and other European countries, specifies a date by giving the ordinal day in the week and the 'calendar week' in a Gregorian year. Section 3.17 of the ISO standard[15] defines a *calendar week* as

> A seven day period within a calendar year, starting on a Monday and identified by its ordinal number within the year; the first calendar week of the year is the one that includes the first Thursday of that year. In the Gregorian calendar, this is equivalent to the week which includes 4 January.

Determining the beginning of the first calendar week of a Gregorian year thus requires determining the Monday on or before January 4 of that year. Since we will need similar determinations for some of the holidays, discussed in a later section, we encapsulate the formula $d - ((d - k) \bmod 7)$ to find the kth day of the week ($k = 0$ for Sunday, and so on) that falls in the seven-day period ending on absolute date d:

```
(defun Kday-on-or-before (date k)
;; Absolute date of the kday on or before date.
;; k = 0 means Sunday, k = 1 means Monday, and so on.
  (- date (mod (- date k) 7)))
```

Applying `Kday-on-or-before` to $d + 6$ gives us the `Kday-on-or-after` an absolute day d. Similarly, applying it to $d + 3$ gives the `Kday-nearest` to absolute date d, applying it to $d - 1$ gives the `Kday-previous` to absolute date d, and applying it to $d + 7$ gives the `Kday-following` absolute date d.

The ISO calendar counts Sunday as the seventh day of the week (throughout this paper we have otherwise counted it as the zeroth day of the week), so we implement this calendar as follows:

```
(defun absolute-from-iso (date)
;; Absolute date equivalent to ISO date = (week day year).
  (let* ((week (first date))
         (day (second date))
         (year (third date)))
;;  Return
    (+ (Kday-on-or-before
```

---

n_{100}	$= \lfloor (d_1 - 1)/36524 \rfloor$	{number of 100 year cycles not included in n_{400}}
d_2	$= (d_1 - 1) \bmod 36524$	{days not included in n_{400} or n_{100}}
n_4	$= \lfloor (d_2 - 1)/1461 \rfloor$	{number of 4 year cycles not included in n_{400} or n_{100}}
d_3	$= (d_2 - 1) \bmod 1461$	{days not included in n_{400}, n_{100}, or n_4}
n_1	$= \lfloor (d_3 - 1)/365 \rfloor$	{number of years not included in n_{400}, n_{100}, or n_4}
d_4	$= (d_3 - 1) \bmod 365$	{days not included in n_{400}, n_{100}, n_4, or n_1}

date is ordinal day d_4 in Gregorian year $400 \times n_{400} + 100 \times n_{100} + 4 \times n_4 + n_1 + 1$

Similar calculations can be used for the Julian, Islamic, or Hebrew calendars.

```
        (absolute-from-gregorian (list 1 4 year))
        1)                        ;;  Days in prior years.
       (* 7 (1- week))            ;;  Days in prior weeks this year.
       (1- day))))                ;;  Prior days this week.

(defun iso-from-absolute (date)
;; ISO (week day year) corresponding to the absolute date.
  (let* ((approx
          (extract-year (gregorian-from-absolute (- date 3))))
         (year (if (>= date
                       (absolute-from-iso (list 1 1 (1+ approx))))
                ;; Then
                   (1+ approx)
                ;; Else
                   approx))
         (week (1+ (quotient
                    (- date (absolute-from-iso (list 1 1 year)))
                    7)))
         (day (if (= 0 (mod date 7))
               ;; Then
                  7
               ;; Else
                  (mod date 7))))
;;  Return
    (list week day year)))
```

THE JULIAN CALENDAR

The calculations for the Julian calendar, which we described in our discussion of the Gregorian calendar, are nearly identical to those for the Gregorian calendar, but we must change the leap-year rule used in determining the last day of a month:

```
(defun last-day-of-julian-month (month year)
;; Last day in Julian month during year.
  (if ;; February in a leap year
      (and (= month 2) (= (mod year 4) 0))
;; Then return
      29
;; Else return
    (nth (1- month) (list 31 28 31 30 31 30 31 31 30 31 30 31))))
```

Converting from a Julian date to an absolute date requires a calculation similar to that in the Gregorian case, but with two minor adjustments: We no longer need consider century-year leap days, but we subtract 2 because absolute date 1 is January 3, 1 C.E. (Julian), and so the first two days of 1 C.E. (Julian) must be excluded.

```
(defun absolute-from-julian (date)
;; Absolute date equivalent to Julian date.
  (let* ((month (extract-month date))
         (year (extract-year date)))
;;  Return
    (+ (extract-day date)    ;; Days so far this month.
       (sum                  ;; Days in prior months this year.
        (last-day-of-julian-month m year) m 1 (< m month))
       (* 365 (1- year))     ;; Days in prior years.
       (quotient (1- year) 4);; Leap days in prior years.
       -2)))                 ;; Days elapsed before absolute date 1.
```

Except for obvious changes in reference from Gregorian to Julian, conversion of absolute dates to Julian dates is identical to conversion of absolute dates to Gregorian dates.

```
(defun julian-from-absolute (date)
;; Julian (month day year) corresponding to absolute date.
  (let*
      ((approx      ;; Approximation from below.
        (quotient (+ date 2) 366))
       (year        ;; Search forward from the approximation.
        (+ approx
           (sum 1 y approx
                (>= date
                    (absolute-from-julian (list 1 1 (1+ y)))))))
       (month       ;; Search forward from January.
        (1+ (sum 1 m 1
                 (> date
                    (absolute-from-julian
                     (list m
                           (last-day-of-julian-month m year)
                           year))))))
       (day         ;; Calculate the day by subtraction.
        (- date (1- (absolute-from-julian (list month 1 year))))))
;;  Return
    (list month day year)))
```

THE ISLAMIC CALENDAR

The Islamic calendar[4–6, 16, 17] is a straightforward, strictly lunar calendar. Its independence of the solar cycle means that its months do not occur in fixed seasons, but migrate through the solar year. Days begin at sunset.

The calendar is computed, by the majority of the Moslem world, starting at sunset of Thursday, July 15, 622 C.E. (Julian), the year of Mohammed's flight to Medina. In essence, Moslems count absolute date 227015 = Friday, July 16, 622 C.E. (Julian) as the beginning of the Islamic year 1, that is, as Muharram 1, 1 A.H. There are 12 Islamic months which contain, alternately, 29 or 30 days:

(1) Muharram	30 days	(7) Rajab	30 days
(2) Safar	29 days	(8) Sha'ban	29 days
(3) Rabi I	30 days	(9) Ramadan	30 days
(4) Rabi II	29 days	(10) Shawwal	29 days
(5) Jumada I	30 days	(11) Dhu al-Qada	30 days
(6) Jumada II	29 days	(12) Dhu al-Hijjah	29 {30} days

The leap-year structure is given in curly brackets—the last month, Dhu al-Hijjah, contains 30 days in the 2nd, 5th, 7th, 10th, 13th, 16th, 18th, 21st, 24th, 26th, and 29th years of a 30-year cycle.* This gives an average month of $29.5305555\cdots$ days. The cycle of common and leap years can be expressed concisely (but obfuscatingly!) by observing that an Islamic year y is a leap year if and only if $(11y + 14) \bmod 30$ is less than 11.

Determining the last day of an Islamic month is thus done by

```
(defun islamic-leap-year (year)
;; True if year is an Islamic leap year.
  (< (mod (+ 14 (* 11 year)) 30) 11))
(defun last-day-of-islamic-month (month year)
;; Last day in month during year on the Islamic calendar.
  (if (or (oddp month)
          (and (= month 12) (islamic-leap-year year)))
;; Then return
      30
;; Else return
    29))
```

The function `oddp` tests for odd integers. (It would be more efficient to compute `islamic-leap-year` by looking up the value in a 30-bit table.)

Converting from an Islamic date to an absolute date is done by summing the days so far in the current month, the days so far in the current Islamic year, the non-leap days in prior Islamic years, the leap days in prior Islamic years, and the days prior to the Islamic calendar.

*A minority of Moslems have a slightly different leap year structure.

```
(defun absolute-from-islamic (date)
;; Absolute date equivalent to Islamic date.
  (let* ((month (extract-month date))
         (year (extract-year date)))
    (+ (extract-day date)     ;; Days so far this month.
       (* 29 (1- month))      ;; Days so far...
       (quotient month 2)     ;;           ...this year.
       (* (1- year) 354)      ;; Non-leap days in prior years.
       (quotient              ;; Leap days in prior years.
        (+ 3 (* 11 year)) 30)
       227014)))              ;; Days before start of calendar.
```

Computing the Islamic date equivalent to a given absolute date is done almost identically to the computations for the Gregorian and Julian calendars: We approximate the year and search linearly for the exact value; then we find the month by linear search and the day of the month by subtraction.

```
(defun islamic-from-absolute (date)
;; Islamic date (month day year) corresponding to absolute date.
  (if ;; Pre-Islamic date.
      (<= date 227014)
;; Then return
      (list 0 0 0)
;; Else
    (let* ((approx        ;; Approximation from below.
            (quotient (- date 227014) 355))
           (year          ;; Search forward from the approximation.
            (+ approx
               (sum 1 y approx
                    (>= date
                        (absolute-from-islamic
                         (list 1 1 (1+ y)))))))
           (month         ;; Search forward from Muharram.
            (1+ (sum 1 m 1
                     (> date
                        (absolute-from-islamic
                         (list m
                               (last-day-of-islamic-month m year)
                               year))))))
           (day           ;; Calculate the day by subtraction.
            (- date (1- (absolute-from-islamic
                         (list month 1 year))))))
      ;; Return
        (list month day year))))
```

It is important to realize that, to some extent, the above calculations are merely hypothetical because there are many disparate forms of the Islamic calendar.[4] Furthermore, much of the Islamic world relies not on such calculations at all, but on proclamation of the new moon by religious authorities. Consequently, the dates given by the Lisp functions here can be in error by a day or two from what will actually be observed in various parts of the Islamic world; this is unavoidable.

THE HEBREW CALENDAR

The Hebrew calendar[4–6, 16, 18–25] promulgated by Hillel II in the mid-fourth century, is by far the most complicated of the five calendars that we consider. Its complexity is inherent in the requirement that calendar months must be strictly lunar while Passover must always occur in the spring. Since the seasons are dependent on the solar year, the Hebrew calendar must harmonize simultaneously with both lunar and solar events. As in the Islamic calendar, days begin at sunset.

The Hebrew year consists of twelve months in a common year and thirteen in a leap year:

(1) Nisan	30 days	(7) Tishri	30 days
(2) Iyyar	29 days	(8) Ḥeshvan	29 or 30 days
(3) Sivan	30 days	(9) Ḳislev	29 or 30 days
(4) Tammuz	29 days	(10) Teveth	29 days
(5) Av	30 days	(11) Shevat	30 days
(6) Elul	29 days	{(12) Adar I	30 days}
		(12) {(13)} Adar {II}	29 days

The leap-year structure is given in curly brackets—in a leap year there is an interpolated twelfth month of 30 days called 'Adar I' to distinguish it from the final month, 'Adar II'. The length of the eighth and ninth months vary from year to year according to criteria that will be explained below. Our ordering of the Hebrew months follows biblical custom (Leviticus 23:5) in which (what was later called) Nisan is the first month. This numbering causes the Hebrew new year (Rosh HaShanah) to begin on the first of Tishri which, by our ordering, is the seventh month—but this too agrees with biblical usage (Leviticus 23:24). Adding up the lengths of the months, we see that a normal year has 353–355 days, whereas a leap year has 383–385 days.

The so-called *Metonic cycle* is based on the observation that 19 mean solar years contain almost exactly 235 lunar months. This correspondence, known to ancient Babylonian astronomers, makes a solar/lunar calendar feasible. The $235 = 12 \times 12 + 7 \times 13$ months in the cycle are divided into twelve years of twelve months and seven years of thirteen months. The Metonic cycle is used in the Hebrew calendar and also for the calculation of Easter (as we discuss in the section on holidays).

In the Hebrew calendar, leap years occur in the 3rd, 6th, 8th, 11th, 14th, 17th, and 19th years of the 19-year cycle. As in the Islamic leap-year structure, this sequence can be computed concisely by noting that Hebrew year y is a leap year if and only if $(7y + 1) \bmod 19$ is less than 7. Thus we determine whether a year is a Hebrew leap year by

```
(defun hebrew-leap-year (year)
;; True if year is a leap year.
  (< (mod (1+ (* 7 year)) 19) 7))
(defun last-month-of-hebrew-year (year)
;; Last month of Hebrew year.
  (if (hebrew-leap-year year)
;; Then return
      13
;; Else return
    12))
```

The number of days in a Hebrew month is a more complex issue. The twelfth month, Adar or Adar I, has 29 days in a common year and 30 days in a leap year, but the numbers of days in the eighth month (Ḥeshvan) and ninth month (Kislev) depend on the overall length of the year, which in turn depends on factors discussed below. Thus we write

```
(defun last-day-of-hebrew-month (month year)
;; Last day of month in Hebrew year.
  (if (or (member month (list 2 4 6 10 13))
          (and (= month 12) (not (hebrew-leap-year year)))
          (and (= month 8) (not (long-heshvan year)))
          (and (= month 9) (short-kislev year)))
;; Then return
      29
;; Else return
    30))
```

where the functions `long-heshvan` and `short-kislev` will be given later.

To present the remainder of the calculations for the Hebrew calendar, it is necessary to describe how Hebrew intervals of time are reckoned. The *day* is divided into 24 *hours* and each hour is divided into 1080 *parts*; a day thus has 25920 parts. For our purposes, it is easier to use just days and parts.

The beginning of the Hebrew new year is determined by the occurrence of the new moon (mean conjunction) of the seventh month (Tishri), subject to possible postponements of a day or two. The average length of a lunar period is taken to be 29 days, 12 hours, and 793 parts, that is, approximately 29.530594 days. The new moon of Tishri, 1 A.M. (the first day of the first year for the Hebrew calendar) is fixed at Sunday night, 5 hours, 204 parts. Thus we calculate the time elapsed from sunset of the preceding Saturday evening, until the new moon of Tishri for the Hebrew year y by computing

$$\text{(1 day, 5604 parts)} + \\ \quad \text{(29 days, 13753 parts)} \times \text{(number of months before year } y\text{)}.$$

The start of each new year, Rosh HaShanah (Tishri 1), coincides with the calculated day of the mean conjunction of Tishri, unless one of four delays is mandated:

- If the mean conjunction is at midday or after, then the new year is delayed.* Sunset is presumed to occur always at 6 p.m.† Since there are then 18 hours from sunset until midday, postponement occurs if the conjunction is at $18 \times 1080 = 19440$ parts or later into the day.

- In no event must the new year be on Sunday, Wednesday, or Friday.‡ This introduces an average 'correction' of about half a day in the calculated time of appearance of the new moon of the month of Tishri.[22]

- To keep the length of a year within the allowable ranges, in rare cases, an additional delaying factor may need to be employed. If Rosh HaShanah were on Tuesday and the conjunction of the following year were at midday or later, then applying the previous two rules would result in delaying the following Rosh HaShanah from Saturday (the day of the next conjunction for a common year) until Monday. This would require an (unacceptable) year length of 356 days, so instead the current Rosh HaShanah is delayed until Thursday.

- Rosh HaShanah on Monday after a leap year can pose a similar problem, by causing the year just ending to be too short. In this case, Rosh HaShanah is delayed until Tuesday.

These rules are perhaps best described algorithmically:

```
(defun hebrew-calendar-elapsed-days (year)
;; Number of days elapsed from the Sunday prior to the start of the
;; Hebrew calendar to the mean conjunction of Tishri of Hebrew year.
  (let*
      ((months-elapsed
        (+
         (* 235            ;; Months in complete cycles so far.
            (quotient (1- year) 19))
         (* 12             ;; Regular months in this cycle.
            (mod (1- year) 19))
         (quotient         ;; Leap months this cycle
          (1+ (* 7 (mod (1- year) 19)))
          19)))
       (parts-elapsed (+ 5604 (* 13753 months-elapsed)))
       (day                                   ;; Conjunction day
        (+ 1 (* 29 months-elapsed) (quotient parts-elapsed 25920)))
       (parts (mod parts-elapsed 25920))      ;; Conjunction parts
```

*In 923 C.E. the calculated conjunction fell just after midday; this caused a short-lived (921–923 C.E.) dispute between Palestinian and Babylonian Jewish authorities about whether this rule should be applied; some scant details can be found in vol. 4, col. 539–540 of the *Encyclopædia Judaica*.[26]

†That is, the moment of sunset is *deemed* 6 p.m. and sunrise is *deemed* 6 a.m., so that the 'daylight hours' and 'nighttime hours' have different lengths that vary according to the seasons. (This is the correct interpretation of chap. 6, par. 2 in Maimonides' code.[22])

‡Excluding Wednesday and Friday prevents Yom Kippur (Tishri 10) from falling on Friday or Sunday; excluding Sunday prevents Hoshanah Rabba (Tishri 21) from falling on Saturday.

```
          (alternative-day
           (if (or
                (>= parts 19440)  ;; If new moon is at or after midday,
                (and
                  (= (mod day 7) 2);; ...or is on a Tuesday...
                  (>= parts 9924)  ;; at 9 hours, 204 parts or later...
                  (not (hebrew-leap-year year)));; of a common year,
                (and
                  (= (mod day 7) 1);; ...or is on a Monday at...
                  (>= parts 16789) ;; 15 hours, 589 parts or later...
                  (hebrew-leap-year;; at the end of a leap year
                   (1- year))))
              ;; Then postpone Rosh HaShanah one day
                (1+ day)
              ;; Else
                day)))
       (if ;; If Rosh HaShanah would occur on Sunday, Wednesday,
           ;; or Friday
           (member (mod alternative-day 7) (list 0 3 5))
     ;; Then postpone it one (more) day and return
           (1+ alternative-day)
     ;; Else return
         alternative-day)))
```

Although the calculations as given above are correct, they involve intermediate values that can be far larger than 2^{24} for current dates. To avoid this problem we can modify the computation of the day and parts of the conjunction in the `let*` to read

```
          ⋮
          (parts-elapsed
           (+ 204
              (* 793 (mod months-elapsed 1080))))
          (hours-elapsed
           (+ 5
              (* 12 months-elapsed)
              (* 793 (quotient months-elapsed 1080))
              (quotient parts-elapsed 1080)))
          (day                                    ;; Conjunction day
           (+ 1
              (* 29 months-elapsed)
              (quotient hours-elapsed 24)))
          (parts                                  ;; Conjunction parts
           (+ (* 1080 (mod hours-elapsed 24))
              (mod parts-elapsed 1080)))
          ⋮
```

With this modification, only 24 bits are necessary for dates in the foreseeable future.

As mentioned above, the length of the year determines the length of the two varying months, Ḥeshvan and Kislev. Ḥeshvan is long (30 days) if the year has 355 or 385 days; Kislev is short (29 days) if the year has 353 or 383 days. The length of the year, in turn, is determined by the dates of the Hebrew new years (Tishri 1) preceding and following the year in question:

```
(defun days-in-hebrew-year (year)
;; Number of days in Hebrew year.
  (- (hebrew-calendar-elapsed-days (1+ year))
     (hebrew-calendar-elapsed-days year)))
(defun long-heshvan (year)
;; True if Heshvan is long in Hebrew year.
  (= (mod (days-in-hebrew-year year) 10) 5))
(defun short-kislev (year)
;; True if Kislev is short in Hebrew year.
  (= (mod (days-in-hebrew-year year) 10) 3))
```

With all the above machinery, we are now ready to convert to and from Hebrew dates, in a manner similar to the previous calendars:

```
(defun absolute-from-hebrew (date)
;; Absolute date of Hebrew date.
  (let* ((month (extract-month date))
         (day (extract-day date))
         (year (extract-year date)))
;; Return
    (+ day                             ;; Days so far this month.
       (if ;; before Tishri
           (< month 7)
     ;; Then add days in prior months this year before and
     ;; after Nisan.
           (+ (sum (last-day-of-hebrew-month m year)
                   m 7 (<= m (last-month-of-hebrew-year year)))
              (sum (last-day-of-hebrew-month m year)
                   m 1 (< m month)))
     ;; Else add days in prior months this year
         (sum (last-day-of-hebrew-month m year) m 7 (< m month)))
     (hebrew-calendar-elapsed-days year);; Days in prior years.
    -1373429)))              ;; Days elapsed before absolute date 1.

(defun hebrew-from-absolute (date)
;; Hebrew (month day year) corresponding to absolute date.
  (let* ((approx    ;; Approximation from below.
          (quotient (+ date 1373429) 366))
         (year      ;; Search forward from the approximation.
          (+ approx (sum 1 y approx
                       (>= date
                           (absolute-from-hebrew
                            (list 7 1 (1+ y)))))))
         (start      ;; Starting month for search for month.
          (if (< date (absolute-from-hebrew (list 1 1 year)))
              ;; Then start at Tishri
```

```
                       7
                 ;; Else start at Nisan
                     1))
            (month      ;; Search forward from either Tishri or Nisan.
             (+ start
                (sum 1 m start
                     (> date
                        (absolute-from-hebrew
                         (list m
                               (last-day-of-hebrew-month m year)
                               year))))))
            (day        ;; Calculate the day by subtraction.
             (- date (1- (absolute-from-hebrew (list month 1 year))))))
    ;; Return
        (list month day year)))
```

The function `hebrew-calendar-elapsed-days` is called repeatedly during the calculations, often several times for the same year. More efficient code would avoid such repetition.

HOLIDAYS

The various calendars are needed to compute the dates of civil and religious holidays. In most of this section, we will take the ethnocentric view that our task is to compute the absolute dates of holidays that occur in a given Gregorian year; there is clearly little difficulty in finding the dates of, say, Islamic holidays in a given Islamic year!

Secular Holidays

Secular holidays on the Gregorian calendar are either on fixed days or on a particular day of the week relative to the beginning or end of a month. (An extensive list of secular holidays can be found in Gregory's *Special Days*.[27]) Fixed holidays are trivial to deal with; for example, to determine the absolute date of American Independence Day in a given Gregorian year we would use

```
(defun independence-day (year)
;; Absolute date of American Independence Day in Gregorian year.
  (absolute-from-gregorian (list 7 4 year)))
```

Other holidays are on the nth occurrence of a given day of the week, counting from either the beginning or the end of the month. American Labor Day, for example, is the first Monday in September, while American Memorial Day is the last Monday in May. We handle such specifications by writing a function that determines the absolute date of the nth kth day in a given month in a given Gregorian year, counting backward from the end of the month when $n < 0$.

```
(defun Nth-Kday (n k month year)
;; Absolute date of the nth kday in Gregorian month, year.
;; If n<0, the nth kday from the end of month is returned
```

```
;; (that is, -1 is the last kday, -2 is the penultimate kday,
;; and so on).  k = 0 means Sunday, k = 1 means Monday, and so on.
  (if (> n 0)
;; Then return
      (+ (Kday-on-or-before            ;; First kday in month.
          (absolute-from-gregorian
           (list month 7 year)) k)
         (* 7 (1- n)))                 ;; Advance n - 1 kdays.
;; Else return
      (+ (Kday-on-or-before            ;; Last kday in month.
          (absolute-from-gregorian
           (list month
                 (last-day-of-gregorian-month month year)
                 year))
          k)
         (* 7 (1+ n)))))               ;; Go back -n - 1 kdays.
```

With this function, we can define holiday dates, such as

```
(defun labor-day (year)
;; Absolute date of American Labor Day in Gregorian year.
  (Nth-Kday 1 1 9 year));; First Monday in September.
(defun memorial-day (year)
;; Absolute date of American Memorial Day in Gregorian year.
  (Nth-Kday -1 1 5 year));; Last Monday in May.
```

or determine the starting and ending dates of American daylight savings time:

```
(defun daylight-savings-start (year)
;; Absolute date of the start of American daylight savings time
;; in Gregorian year.
  (Nth-Kday 1 0 4 year));; First Sunday in April.
(defun daylight-savings-end (year)
;; Absolute date of the end of American daylight savings time
;; in Gregorian year.
  (Nth-Kday -1 0 10 year));; Last Sunday in October.
```

Christian Holidays

The main Christian holidays are Christmas, Easter, and various days connected with them (Advent, Ash Wednesday, Good Friday, and others; see vol. V, pp. 844–853 of the *Encyclopædia of Religion and Ethics.*[7]) In addition to the complicated calculations necessary to determine the date of Easter, there are complications that result from the Eastern Orthodox practice of celebrating Christmas according to the Julian calendar.

The date of Christmas on the Gregorian calendar is fixed and presents no problem:

```
(defun christmas (year)
;; Absolute date of Christmas in Gregorian year.
  (absolute-from-gregorian (list 12 25 year)))
```

The related dates of Advent (Sunday closest to November 30; this is equivalent to the Sunday on or before December 3) and Epiphany (twelve days after Christmas) are computed by

```
(defun advent (year)
;; Absolute date of Advent in Gregorian year.
  (Kday-on-or-before (absolute-from-gregorian (list 12 3 year)) 0))
(defun epiphany (year)
;; Absolute date of Epiphany in Gregorian year.
  (+ 12 (christmas year)))
```

The date of Assumption (August 15), celebrated in Catholic and Eastern Orthodox countries, is fixed, and presents no problem.

The date of Eastern Orthodox Christmas occurring in a given Gregorian year is more involved. Since the Julian year is always at least as long as the corresponding Gregorian year, Eastern Orthodox Christmas can occur at most once in a given Gregorian year, but it can occur either at the beginning or the end; in some years (like 1100 C.E.) it does not occur at all.

```
(defun eastern-orthodox-christmas (year)
;; List of zero or one absolute dates of Eastern Orthodox
;; Christmas in Gregorian year.
  (let* ((jan1 (absolute-from-gregorian (list 1 1 year)))
         (dec31 (absolute-from-gregorian (list 12 31 year)))
         (y (extract-year (julian-from-absolute jan1)))
         (c1 (absolute-from-julian (list 12 25 y)))
         (c2 (absolute-from-julian (list 12 25 (1+ y)))))
   (append
    (if ;; c1 occurs in current year
        (<= jan1 c1 dec31)
;;   Then that date; otherwise, none
        (list c1) nil)
    (if ;; c2 occurs in current year
        (<= jan1 c2 dec31)
;;   Then that date; otherwise, none
        (list c2) nil))))
```

The function `append` concatenates its arguments into one list.

The calculation of the date of Easter has a fascinating history.[11, 12] Algorithms and computer programs abound[5, 6, 28–32] (the discussion of the mathematical aspects of the calculation in O'Beirne's *Puzzles and Paradoxes*[31] is especially nice), but we include Lisp functions here for completeness and because our absolute-date approach allows considerable simplification of 'classical' algorithms.

The date of Easter was fixed in 325 C.E. by the Council of Nicæa to be the first Sunday after the first full moon occurring on or after the vernal equinox. This definition seems precise, but in reality accurate determination of the full moon and the vernal equinox is quite complex and simpler approximations are used in practice; as Kepler declared,[12] 'Easter is a feast, not a planet'. The date of Easter is thus based on the presumption that the vernal equinox is always March 21 and on approximations to the lunar phases called *epacts*.

Before the Gregorian reform of the Julian calendar the approximations were fairly crude. Assuming the Metonic cycle were accurate would mean that the phase of the moon on January 1 would be the same every 19 years. Hence, the epact can be approximated by multiplying the number of years since the start of the current Metonic cycle (called the 'golden number') by the eleven-day difference between a common year of 365 days and 12 lunar months of 29.5 days and adjusting by the epact of January 1, 1 C.E.—all this done modulo 30. From the epact one can calculate the phase of the moon (in days) on April 5 (ignoring February 29) and go back that many days from April 19, giving a date between March 21 and April 19 (inclusive) for the (ecclesiastical) 'Paschal full moon'. Thus the equivalent of the following calculation was used to determine Easter from 325 C.E. until the adoption of the Gregorian calendar:[5, 6, 29]

```
(defun nicaean-rule-easter (year)
;; Absolute date of Easter in Julian year, according to the rule
;; of the Council of Nicaea.
  (let* ((shifted-epact ;; Age of moon for April 5.
          (mod (+ 14
                  (* 11 (mod year 19)))
               30))
         (paschal-moon  ;; Day after full moon on or after March 21.
          (- (absolute-from-julian (list 4 19 year))
             shifted-epact)))
;; Return the Sunday following the Paschal moon
    (Kday-on-or-before (+ paschal-moon 7) 0)))
```

The Gregorian reform included, for the calculation of Easter, a far more accurate approximation to the lunar phases due to Clavius. Three corrections are employed:

- The Gregorian change in the calendar (three out of four century years are common years) is taken into account in the calculation of epacts.
- About every 222 years, a one-day correction is made to compensate for the inaccuracy of the Metonic cycle.

- There are approximately 29.5 days between successive new moons, while the lunar month in which Easter occurs is always taken to have 29 days, so an additional adjustment to the epact (which can take on 30 values) is sometimes needed.*

This is the method now used:[5, 6, 30]

```
(defun easter (year)
;; Absolute date of Easter in Gregorian year.
  (let* ((century (1+ (quotient year 100)))
         (shifted-epact       ;; Age of moon for April 5...
          (mod
           (+ 14 (* 11 (mod year 19));;     ...by Nicaean rule
              (-      ;; ...corrected for the Gregorian century rule
               (quotient (* 3 century) 4))
              (quotient;; ...corrected for Metonic cycle inaccuracy.
               (+ 5 (* 8 century)) 25)
              (* 30 century));;             Keeps value positive.
           30))
         (adjusted-epact      ;;  Adjust for 29.5 day month.
          (if (or (= shifted-epact 0)
                  (and (= shifted-epact 1) (< 10 (mod year 19))))
        ;; Then
              (1+ shifted-epact)
        ;; Else
            shifted-epact))
         (paschal-moon;; Day after full moon on or after March 21.
          (- (absolute-from-gregorian (list 4 19 year))
             adjusted-epact)))
;; Return the Sunday following the Paschal moon.
    (Kday-on-or-before (+ paschal-moon 7) 0)))
```

Many Christian holidays depend on the date of Easter: Septuagesima Sunday (63 days before), Sexagesima Sunday (56 days before), Shrove Sunday (49 days before), Shrove Monday (48 days before), Shrove Tuesday (47 days before), Ash Wednesday (46 days before), Passion Sunday (14 days before), Palm Sunday (seven days before), Maundy Thursday (three days before), Good Friday (two days before), Rogation Sunday (35 days after), Ascension Day (39 days after), Pentecost (also called Whitsunday—49 days after), Whitmundy (50 days after), Trinity Sunday (56 days after), and Corpus Christi (60 days after). All these are easily computed; for example

```
(defun pentecost (year)
;; Absolute date of Pentecost in Gregorian year.
  (+ 49 (easter year)))
```

*The details of this adjustment were designed by Clavius to keep Easter within the same range of dates as in the Julian calendar, March 22 through April 25.[31]

Islamic Holidays

Determining the absolute dates of Islamic holidays occurring in a given Gregorian year is complicated because an Islamic year is always shorter than the Gregorian year, so each Gregorian year contains parts of at least two and sometimes three successive Islamic years. Hence any given Islamic date occurs at least once and possibly twice in any given Gregorian year. For example, Islamic New Year (Muharram 1) occurred twice in 1943: on January 8 and again on December 28. Accordingly, we will approach the problem of the Islamic holidays by writing a general function to return a list of the absolute dates of a given Islamic date occurring in a given Gregorian year:

```
(defun islamic-date (month day year)
;; List of the absolute dates of Islamic month, day
;; that occur in Gregorian year.
  (let* ((jan1 (absolute-from-gregorian (list 1 1 year)))
         (dec31 (absolute-from-gregorian (list 12 31 year)))
         (y (extract-year (islamic-from-absolute jan1)))
;; The possible occurrences in one year are
         (date1 (absolute-from-islamic (list month day y)))
         (date2 (absolute-from-islamic (list month day (1+ y))))
         (date3 (absolute-from-islamic (list month day (+ 2 y)))))
;; Combine in one list those that occur in current year
    (append
      (if (<= jan1 date1 dec31)
          (list date1) nil)
      (if (<= jan1 date2 dec31)
          (list date2) nil)
      (if (<= jan1 date3 dec31)
          (list date3) nil))))
```

There is little uniformity among the Islamic sects and countries as to holidays. In general, the principal holidays of the Islamic year are Islamic New Year (Muharram 1), Ashura (Muharram 10), Mulad-al-Nabi (Rabi I 12), Shab-e-Mi'raj (Rajab 26), Shab-e-Bara't (Sha'ban 15), Ramadan (Ramadan 1), Shab-e Qadr (Ramadan 27), Id-al-Fitr (Shawwal 1), and Id-al-Adha (Dhu-al-Hijjah 10).* Like all Islamic days, an Islamic holiday begins at sunset the prior evening. We can determine a list of the corresponding absolute dates of occurrence in a given Gregorian year by using `islamic-date` above, as in

```
(defun mulad-al-nabi (year)
;; List of absolute dates of Mulad-al-Nabi occurring in
;; Gregorian year.
  (islamic-date 3 12 year))
```

Islamic holidays begin at sunset on the prior evening. It bears reiterating that the determination of the Islamic holidays cannot be fully accurate since the precise day of their occurrence is dependent on proclamation by religious authorities.

---

*Other days, too, have religious significance, for example, the entire month of Ramadan.

Jewish Holidays

As throughout this section, we consider our problem to be the determination of the Jewish holidays that occur in a specified Gregorian year. Since the Hebrew year is, on the average, consistently aligned with the Gregorian year, each Jewish holiday occurs just once in a given Gregorian year (with a minor exception noted below). The major holidays of the Hebrew year occur on fixed days on the Hebrew calendar, but occur only in fixed seasons on the Gregorian calendar.[7, 24] They are easy to determine on the Gregorian calendar with the machinery developed above, provided we observe that the Hebrew year that began in the fall of 1 C.E. (Gregorian) was 3762 A.M., so we have the relation

$$y + 3761 = \text{Hebrew new year occurring in the fall of Gregorian year } y.$$

This means that holidays occurring in the fall and early winter of the Gregorian year y occur in the Hebrew year $y + 3761$, while holidays in the late winter, spring, and summer occur in Hebrew year $y + 3760$. For example, to find the absolute date of Yom Kippur (Tishri 10) in a Gregorian year, we would use

```
(defun yom-kippur (year)
;; Absolute date of Yom Kippur occurring in Gregorian year.
  (absolute-from-hebrew (list 7 10 (+ year 3761))))
```

The absolute dates of Rosh HaShanah (Tishri 1), Sukkot (Tishri 15), Hoshanah Rabba (Tishri 21), Shemini Aẓereth (Tishri 22), and Simḥat Torah (Tishri 23, outside Israel) are identically determined. (See p. 800 of *Winning Ways, Volume 2: Games in Particular*[28] for another way to determine the date of Rosh HaShanah.) As on the Islamic calendar, all Hebrew holidays begin at sunset the prior evening.

The dates of the other major holidays, Passover (Nisan 15), ending of Passover (Nisan 21), and Shavuot (Sivan 6), are determined similarly, but since these holidays occur in the spring, the year corresponding to Gregorian year y is $y + 3760$.* Thus, for example, we determine the absolute date of Passover by

```
(defun passover (year)
;; Absolute date of Passover occurring in Gregorian year.
  (absolute-from-hebrew (list 1 15 (+ year 3760))))
```

(Gauss developed an interesting formula to determine the Gregorian date of Passover in a given year.[20])

The minor holidays of the Hebrew year are the 'intermediate' days of Sukkot (Tishri 16–21) and Passover (Nisan 16–20), Ḥanukkah (eight days, beginning on Kislev 25), Tu-B'Shevat (Shevat 15), and Purim (Adar 14 in normal years, Adar II 14 in leap years). Ḥanukkah occurs in late fall/early winter, so Ḥanukkah of Gregorian year y occurs in the Hebrew year $y + 3761$, while Tu-B'Shevat occurs in late winter/early spring and hence Tu-B'Shevat of Gregorian year y occurs in Hebrew year $y+3760$; thus

*Conservative and Orthodox Jews observe two days Rosh HaShanah—Tishri 1 and 2. Outside Israel, they also observe Tishri 16, Nisan 16, Nisan 22, and Sivan 7 as holidays.

these two holidays are handled as were Rosh HaShanah and Passover, respectively. Purim always occurs in late winter or early spring, so its absolute date is computed by

```
(defun purim (year)
;; Absolute date of Purim occurring in Gregorian year.
  (absolute-from-hebrew
    (list
      (last-month-of-hebrew-year (+ year 3760));; Adar or Adar II
      14
      (+ year 3760))))
```

The Hebrew year contains several fast days that, though specified by particular Hebrew calendar dates, are shifted when those days occur on Saturday. The fast days are Tzom Gedaliah (Tishri 3), Tzom Teveth (Teveth 10), Ta'anit Esther (the day before Purim), Tzom Tammuz (Tammuz 17), and Tisha B'Av (Av 9). When Purim is on Sunday, Ta'anith Esther occurs on the preceding Thursday, so we can write

```
(defun ta-anit-esther (year)
;; Absolute date of Purim occurring in Gregorian year.
  (let* ((purim-date (purim year)))
    (if ;; Purim is on Sunday
        (= (mod purim-date 7) 0)
  ;; Then return prior Thursday
        (- purim-date 3)
  ;; Else return previous day
      (1- purim-date))))
```

Each of the other fast days, as well as Shushan Purim (the day after Purim, celebrated in Jerusalem), is postponed to the following day (Sunday) when it occurs on Saturday. Since Tzom Gedaliah is always in the fall, while Tzom Tammuz and Tisha B'Av are always in the summer, their determination is easy. For example,

```
(defun tisha-b-av (year)
;; Absolute date of Tisha B'Av occurring in Gregorian year.
  (let* ((ninth-of-av
          (absolute-from-hebrew (list 5 9 (+ year 3760)))))
    (if ;; Ninth of Av is Saturday
        (= (mod ninth-of-av 7) 6)
  ;; Then return the next day
        (1+ ninth-of-av)
  ;; Else return
      ninth-of-av)))
```

Tzom Teveth, which can never occur on Saturday, should be handled like Islamic holidays because Teveth 10 can fall on either side of January 1, so a single Gregorian calendar year can have zero, one, or two occurrences of Tzom Teveth. For example, Tzom Teveth occurred twice in 1982, but not at all in 1984. We leave it to the reader to work out the details.

On the Hebrew calendar, the first day of each month except Tishri is called Rosh Ḥodesh and has minor ritual significance. When the preceding month has 30 days, Rosh Ḥodesh includes also the last day of the preceding month. The determination of these days is elementary. Some other dates of significance depend on the Julian approximation of the tropical year in which each of the four seasons is taken to be 91.3125 days long.*

Finally, the Hebrew calendar contains what we might term 'personal' days: One's birthday according to the Hebrew calendar determines the day of one's *Bat* (for girls) or *Bar* (for boys) *Mitzvah* (the twelfth or thirteenth birthday). Dates of death determine when *Kaddish* is recited (*yahrzeits*) for parents (and sometimes for other relatives). These are ordinarily just anniversary dates, but the leap year structure and the varying number of days in some months require that alternative days be used in certain years, just as someone born February 29 on the Gregorian calendar has to substitute an alternate day in common years.

The particular alternatives are best described algorithmically; since the problem here is the determination of a date in a given Hebrew year, we write the functions in that way:†

```
(defun hebrew-birthday (birthdate year)
;; Absolute date of the anniversary of Hebrew birthdate
;; occurring in Hebrew year.
  (let* ((birth-day (extract-day birthdate))
         (birth-month (extract-month birthdate))
         (birth-year (extract-year birthdate)))
    (if ;; It's Adar in a normal year or Adar II in a leap year,
        (= birth-month (last-month-of-hebrew-year birth-year))
   ;; Then use the same day in last month of year.
        (absolute-from-hebrew
         (list (last-month-of-hebrew-year year) birth-day year))
   ;; Else use the normal anniversary of the birth date,
   ;; or the corresponding day in years without that date
      (absolute-from-hebrew (list birth-month birth-day year)))))
```

This code takes advantage of the fact that `absolute-from-hebrew` works for dates (*month i year*) and always returns the absolute date of the $(i-1)$st day after (*month* 1 *year*), even if the month has fewer than i days.

```
(defun yahrzeit (death-date year)
;; Absolute date of the anniversary of Hebrew death-date
;; occurring in Hebrew year.
```

*By one traditional Hebrew reckoning, the vernal equinox of year 5685 A.M. was at 6 p.m. Wednesday evening, March 26, 1925 C.E. (Julian). It recurs on that day of the Julian calendar and at that hour of the week every 28 years (*birkhat haḥama*). The beginning of *sh'ela* (request for rain) outside Israel in Julian year y (meant to correspond to 60 days after the autumnal equinox) is absolute day `(- (absolute-from-julian (list 3 26 (1+ y))) 124)`. See Spier's *The Comprehensive Hebrew Calendar*.[24]

†There are minor variations in custom regarding the date of *yahrzeit* in some of these cases, but they need not concern us here.

```
(let* ((death-day (extract-day death-date))
       (death-month (extract-month death-date))
       (death-year (extract-year death-date)))
  (cond
   ;; If it's Heshvan 30 it depends on the first anniversary; if
   ;; that was not Heshvan 30, use the day before Kislev 1.
   ((and (= death-month 8)
         (= death-day 30)
         (not (long-heshvan (1+ death-year))))
    (1- (absolute-from-hebrew (list 9 1 year))))
   ;; If it's Kislev 30 it depends on the first anniversary; if
   ;; that was not Kislev 30, use the day before Teveth 1.
   ((and (= death-month 9)
         (= death-day 30)
         (short-kislev (1+ death-year)))
    (1- (absolute-from-hebrew (list 10 1 year))))
   ;; If it's Adar II, use the same day in last month of
   ;; year (Adar or Adar II).
   ((= death-month 13)
    (absolute-from-hebrew
     (list (last-month-of-hebrew-year year) death-day year)))
   ;; If it's the 30th in Adar I and year is not a leap year
   ;; (so Adar has only 29 days), use the last day in Shevat.
   ((and (= death-day 30)
         (= death-month 12)
         (not (hebrew-leap-year death-year)))
    (absolute-from-hebrew (list 11 30 year)))
   ;; In all other cases, use the normal anniversary of the
   ;; date of death.
   (t (absolute-from-hebrew
       (list death-month death-day year))))))
```

The `cond` statement lists a sequence of tests and values, like a generalized case statement.

OTHER CALENDARS

In this section, we briefly describe two calendars for which we would have liked to provide algorithms.

Hindu Calendars

The Hindu calendar is a luni-solar calendar based on approximations to the true astronomical day, synodic month, and sidereal year, rather than to their mean values, as is the case for the calendars considered in previous sections. There are three main variations in use in India; the one based on the *Sūrya-Siddhānta* (circa 1000 C.E.) uses a value of $365.258756481481\cdots$ days for the length of the sidereal year. Days of a month are numbered according to the computed phase of the moon at sunrise (in a particular location). Months are named according to the position of the sun in the zodiac at the computed time of new (or, in some variations, full) moon (itself a function of the sun's position). Since the true solar and lunar motions across the celestial sphere vary in speed, this method of reckoning leads occasionally to consecutive days bearing the *same* ordinal number, to skipped numbers, and, similarly, to repeated and lost months.[33, 34]

The necessary computational mechanisms for handling the Hindu calendars are far too complex for inclusion in this paper.[35] Indeed, as van Wijk wrote,[34] 'The rules the Sūrya-Siddhānta gives for calculating the time of true sunrise are exceedingly complicated, and inapplicable in practice.'

The Chinese Calendar

The Chinese year consists of twelve lunar months in a common year and thirteen lunar months in a leap year. Chinese New Year occurs at the beginning of the lunar month during which the sun's tropical longitude is 330 degrees.[36]

The Chinese calendar is similar to the Hindu in its use of true values for solar and lunar events. Unlike the Hindu calendars, the Chinese calendar makers use modern astronomy to decide close calls (whether the sun and moon at the point of true conjunction are in one constellation or in another). The formulae of the ephemeris[9] can be used for this purpose, but require floating point arithmetic, and are not valid over extended periods of time. For these reasons, we have not included Lisp functions for the Chinese calendar or the determination of Chinese New Year in this paper.

ACKNOWLEDGEMENTS

We thank Yaacov Choueka, Douglas Comer, Supyan Hussin, Mitch Medow, Lester Penner, and Daniel Schindler for their comments.

REFERENCES

Note: Only English language references are included in the following list. Some of these are relatively inaccessible.

1. R. M. Stallman, *GNU Emacs Manual*, 6th ed., Free Software Foundation, Cambridge, MA, 1986.
2. G. L. Steele, Jr., COMMON LISP: *The Language*, Digital Press, Boston, MA, 1984.
3. F. Parise, *The Book of Calendars*, Facts on File, New York, 1982.
4. V. V. Tsybulsky, *Calendars of Middle East Countries*, Institute of Oriental Studies, USSR Academy of Sciences, Moscow, 1979.
5. W. S. B. Woolhouse, *Measures, Weights, & Moneys of All Nations: And an Analysis of the Christian, Hebrew, and Mahometan Calendars*, 6th ed., Crosby Lockwood, London, 1881.
6. W. S. B. Woolhouse, 'Calendar', *The Encyclopædia Britannica*, 11th ed., vol. 4, pp. 987–1004, The Encyclopædia Britannica Co., New York, 1910. The same article also appears in the eighth through tenth editions.
7. J. Hastings, ed., *Encyclopædia of Religion and Ethics*, Charles Scribner's Sons, New York, 1911.
8. L. Lamport, 'On the proof of correctness of a calendar program', *Comm. ACM*, **22**, 554–556 (1979).
9. *Explanatory Supplement to the Astronomical Ephemeris and the American Ephemeris and Nautical Almanac*, Her Majesty's Stationery Office, London, 1961.
10. G. V. Coyne, M. A. Hoskin and O. Pedersen, *Gregorian Reform of the Calendar: Proceedings of the Vatican Conference to Commemorate Its 400th Anniversary, 1582–1982*, Pontifica Academica Scientiarum, Specola Vaticana, Vatican, 1983.
11. J. Dutka, 'On the Gregorian revision of the Julian calendar', *Math. Intelligencer*, **10**, 56–64 (1988).
12. G. Moyer, 'The Gregorian calendar', *Sci. Amer.*, **246**, (5), 144–152 (May 1982).
13. V. F. Rickey, 'Mathematics of the Gregorian calendar', *Math. Intelligencer*, **7**, 53–56 (1985).
14. E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*, Prentice-Hall, Englewood Cliffs, NJ, 1977.
15. *Data Elements and Interchange Formats—Information Interchange—Representation of Dates*, ISO 8601, International Organization for Standardization, 1988. This standard replaced ISO 2015, the original document describing the ISO calendar.
16. S. B. Burnaby, *Elements of the Jewish and Muhammadan Calendars*, George Bell and Sons, London, 1901.
17. G. S. P. Freeman-Grenville, *The Muslim and Christian Calendars*, 2nd ed., Rex Collings, London, 1977.
18. N. Bushwick, *Understanding the Jewish Calendar*, Moznaim Publishing Corp., New York, 1989.
19. W. M. Feldman, *Rabbinical Mathematics and Astronomy*, M. L. Cailingold, London, 1931; 3rd corrected ed., Sepher-Hermon, New York, 1978.
20. M. Friedländer, 'Calendar', *The Jewish Encyclopedia*, Funk and Wagnalls, New York, 1906, pp. 501–508.
21. L. Levi, *Jewish Chronomomy: The Calendar and Times of Day in Jewish Law*, Gur Aryeh Institute for Advanced Jewish Scholarship, Brooklyn, NY, 1967.
22. Maimonides (= Moshe ben Maimon), *Mishneh Torah: Sefer Zemanim—Hilḥot Kiddush HaḤodesh*, 1177. Translated by S. Gandz (with commentary by J. Obermann and O. Neugebauer), as *Code of Maimonides, Book Three, Treatise Eight, Sanctification of the New Moon*, Yale Judaica Series, vol. XI, Yale University Press, New Haven, CT, 1956. Addenda and corrigenda by E. J. Wiesenberg appear at the end of *Code of Maimonides, Book Three, The Book of Seasons*, translated by S. Gandz and H. Klein, Yale Judaica Series, vol. XIV, Yale University Press, New Haven, CT, 1961.

23. L. A. Resnikoff, 'Jewish calendar calculations', *Scripta Math.*, **9**, 191–195, 274–277 (1943).
24. A. Spier, *The Comprehensive Hebrew Calendar*, 3rd ed., Feldheim Publishers, New York, 1986.
25. E. J. Wiesenberg, 'Calendar', *Encyclopædia Judaica*, vol. 5, col. 43–50, Macmillan, New York, 1971.
26. C. Roth, ed., *Encyclopædia Judaica*, Macmillan, New York, 1971.
27. R. W. Gregory, *Special Days*, Citadel, Secaucus, NJ, 1975. Previous editions appeared under the title, *Anniversaries and Holidays.*
28. E. R. Berlekamp, J. H. Conway and R. K. Guy, *Winning Ways: Volume 2, Games in Particular*, Academic Press, New York, 1982.
29. D. E. Knuth, 'The calculation of Easter', *Comm. ACM*, **5**, 209–210 (1962).
30. D. E. Knuth, *The Art of Computer Programming, Volume I: Fundamental Algorithms*, 2nd ed., Addison-Wesley, Reading, MA, 1973.
31. T. H. O'Beirne, *Puzzles and Paradoxes*, Oxford University Press, London, 1965. Reprinted by Dover Publications, New York, 1984.
32. J. V. Uspensky and M. A. Heaslet, *Elementary Number Theory*, McGraw-Hill, New York, 1939.
33. V. B. Ketkar, 'Indian and foreign chronology: with theory, practice and tables, B.C. 3102 to 2100 A.D. and notices of the Vedic, the ancient Indian, the Chinese, the Jewish, the ecclesiastical and the Coptic calendars', *J. Royal Asiatic Soc., Bombay Branch*, **XXVI**, (LXXV-A, *Extra Number*), 1923.
34. W. E. van Wijk, *Decimal Tables for the Reduction of Hindu Dates from the Data of the Sūrya-Siddhānta*, Martinus Nijhoff, The Hague, 1938.
35. W. E. van Wijk, 'On Hindu chronology V: Decimal tables for calculating true local time, according to the Sūrya-Siddhānta', *Acta Orientalia*, **V**, 1–27 (1926).
36. H. Fritsche, *On Chronology and the Construction of the Calendar with Special Regard to the Chinese Computation of Time Compared with the European*, R. Laverentz, St. Petersburg, 1886.