Implementing Solar Astronomical Calendars

Nachum Dershowitz
Edward M. Reingold

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 West Springfield Avenue
Urbana, Illinois 61801-2987, USA

May 29, 2001

In this note we describe a unified implementation of calendars whose year is based on the astronomical solar cycle—that is, on the precise solar longitude at a specifed time. For example, the astronomical Persian calendar begins its new year on the day when the vernal equinox (approximately March 21) occurs before apparent noon (the middle point of the day, not clock time) and is postponed to the next day if the equinox is after noon. Other calendars of this type include the French Revolutionary calendar and the future form of the Bahá'í calendar. Our approach also offers a slight simplification to the implementation of the Chinese lunisolar calendar.

Our descriptions here build on, and assume familiarity with, the collection of (Common) Lisp functions given in Appendix B of our book, *Calendrical Calculations*, Cambridge University Press, 1997. For each calendar considered there, a date is converted to and from a *fixed date* (or R.D.), which is a day numbering starting with 1 January 1 (Gregorian) as R.D. 1.

We begin by giving, in the next section, some general support functions; we follow that with a section describing a simple astronomical solar calendar that allows to describe such implementations in general. Then we show how the necessary computations for the French Revolutionary, the astronomical Persian, and the future Bahá'í calendars follow easily.

Some Support Functions

First, we need a function to tell us directly the solar longitude at a given moment (instead of at a given Julian day number, as calculated by the function `solar-longitude` given in *Calendrical Calculations*):

```
(defun solar-longitude-at-moment (moment)
  ;; TYPE moment -> angle
  ;; Longitude of sun at moment.
  (solar-longitude (jd-from-moment moment)))
```

We also need to convert back and forth between local (clock) time and apparent (sundial) time expressed in *moments*, that is, days and fraction of day since R.D. 0 (and not in Julian day numbers as used for the astronomical calculations in *Calendrical Calculations*):

```
(defun local-from-apparent-moment (moment)
  ;; TYPE moment -> moment
  ;; Local time from sundial time.
  (- moment (equation-of-time (jd-from-moment moment))))
```

```
(defun apparent-from-local-moment (moment)
  ;; TYPE moment -> moment
  ;; Sundial time at local time.
  (+ moment (equation-of-time (jd-from-moment moment))))
```

The functions `equation-of-time` and `jd-from-moment` are defined in *Calendrical Calculations* and have the obvious effect.

Finally, we will find it convenient to employ a Common Lisp macro for searching for the smallest integer after some `initial` value for which a given `condition` holds:

```
(defmacro next (index initial condition)
  ;; TYPE (* integer (integer->boolean)) -> integer
  ;; First integer greater or equal to initial such that
  ;; condition holds.
  `(do ((,index ,initial (1+ ,index)))
       (,condition ,index)))
```

This macro is a simplified version of the `sum` macro in *Calendrical Calculations*.

Astronomical Functions

The key to implementing any astronomical solar calendar is to determine the day of the most recent new year on or before a given fixed date. We assume that the new year begins on the day when the solar longitude was at least `l` at a *critical moment* which is defined for every fixed date `d` by a function `critical-moment`. The longitude `l` need not actually be that of the new year; the following macro simply finds the fixed date at the critical moment of which the longitude reaches `l`:

```
(defmacro solar-new-year-on-or-before (date l d critical-moment)
  ;; TYPE fixed-date angle * (fixed-date -> moment) -> fixed-date
  ;; Last fixed date on or before date when solar longitude
  ;; was at least l at critical-moment (UT) of that date.
  (let* ((theta (gensym))
         (start (gensym)))
    `(let* ((,d ,date)
            (,theta ;; Longitude at critical moment on given date.
             (solar-longitude-at-moment ,critical-moment))
            (,start ;; Date before prior occurrence of l
             (- ,date 5
                ;; Approximate number of days since longitude was l.
                (floor (* mean-tropical-year 1/360
                          (degrees (- ,theta ,l)))))))
       (next ,d ,start (<= ,l
                           (solar-longitude-at-moment ,critical-moment)
                           (+ 2 ,l))))))
```

First we determine the solar longitude at the critical moment of the given fixed date. We use this value to tell us approximately how far back we must go before `date` to reach the desired longitude `l`; for safety (since the apparent speed of the sun varies) we go back an additional five days. (The function `degrees` normalizes an angle to the range 0..360.) Then we search forward with `next`, day by day, from `start`, until we find the date on which the solar longitude has reached `l` by the critical moment. The variable `d` is just the formal parameter of the function `critical-moment`.

Given an epoch for a solar astronomical calendar (that is, the fixed date of the first day of the first year of that calendar), we want to be able to determine the first fixed date y `years` after epoch with solar longitude at least `l` at the critical moment of that date. We assume that the integral longitude preceding the critical moment of the epoch day is the longitude that determines the new year. We do this as follows:

```
(defmacro solar-event (epoch years l d critical-moment)
  ;; TYPE fixed-date integer angle * (fixed-date -> moment)
  ;; -> fixed-date
  ;; First fixed date years solar years after epoch with solar
  ;; longitude at least l at critical-moment of that date.
  ;; It is assumed that the integral longitude preceding the
  ;; critical-moment of the epoch day is the longitude that
  ;; determines the new year.
  (let* ((epochal-longitude (gensym)))
    `(let* ((,d ,epoch)
            (,epochal-longitude
             (floor (solar-longitude-at-moment
                     ,critical-moment))))
       (solar-new-year-on-or-before
        (floor (+ ,epoch 10
                  (* mean-tropical-year ,years)
                  (* mean-tropical-year 1/360
                     (degrees (- ,l ,epochal-longitude)))))
        ,l ,d ,critical-moment))))
```

We find the epochal longitude (the integral solar longitude during the day of the epoch) and compare it to the desired longitude `l`. This tells us approximately what fraction of a year past the new year are needed for the sun to return to longitude `l`. Adding y mean tropical `years` and that fractional year to the epoch, plus ten days as a safety factor, gives a fixed date d following the desired solar event. The macro `solar-new-year-on-or-before`, given d, then finds the date when the solar longitude already reached `l` at the `critical-moment`.

A Simple Example: The Urbana Astrological Calendar

In this section we describe a simple, hypothetical solar astronomical calendar that we call the "Urbana Astrological" calendar. We begin with a constructor for dates on this calendar; each date is a triple:

```
(defun urbana-astrological-date (month day year)
  ;; TYPE (urbana-astrological-month urbana-astrological-day
  ;; TYPE  urbana-astrological-year) -> urbana-astrological-date
  (list month day year))
```

The new year on this imaginary calendar begins on the day after the winter solstice in Urbana, Illinois (we use the function `urbana-winter` from *Calendrical Calculations*). The epoch is this event in Gregorian year 2000, 730,476 R.D. = 22 December 2000 (Gregorian), which is the start of year 0 on this calendar:

```
(defconstant urbana-astrological-epoch
  ;; TYPE fixed-date
  ;; Winter solstice in the year 2000.
  (ceiling (urbana-winter 2000)))
```

The calendar has twelve months, each a true solar month of 30° solar longitude, corresponding to a zodiacal constellation. Each month comprises between 29 and 32 days, depending on the precise time for the sun to move through the required 30°. The critical moment is midnight standard time at the start of the day in question. The following function converts local midnight in Urbana (360 minutes behind Greenwich) to U.T.:

```
(defconstant urbana-time-zone
  ;; TYPE minute
```

```
  ;; The difference (in minutes) of Central time zone
  ;; from Universal Time.
  -360)

(defun midnight-in-urbana (date)
  ;; TYPE fixed-date -> moment
  ;; Moment in U.T. when it's midnight in Urbana.
  (universal-from-local date urbana-time-zone))
```

The function `universal-from-local` converts local time to U.T. time.

To determine a fixed date from an Urbana astrological date we use the year of astrological date to determine the number of elapsed solar years; we use the month of the astrological date to determine the solar longitude at the start of that month. With these two values, the function `solar-event` can tell us the fixed date of occurrence of entry into that constellation in Urbana, to which we add the day of the astrological date minus one:

```
(defun fixed-from-urbana-astrological (u-date)
  ;; TYPE astrological-date -> fixed-date
  ;; Fixed date of Urbana astrological date.
  (let* ((month (standard-month u-date))
         (day (standard-day u-date))
         (year (standard-year u-date)))
    (+ day -1
       (solar-event urbana-astrological-epoch year
                    (degrees (+ 270 (* (1- month) 30)))
                    x (midnight-in-urbana x)))))
```

To invert the process and convert a fixed date to an Urbana astrological date, we find the previous new year (day of the solstice in Urbana) and use that to find the year number by dividing the number of elapsed days since the epoch by the mean length of a year, compute the month number from the solar longitude on that fixed date (counting one month for each 30° past the longitude of 270° of the solstice), and compute the day of the month by subtracting the given fixed date from the start of the month:

```
(defun urbana-astrological-from-fixed (date)
  ;; TYPE fixed-date -> astrological-date
  ;; Urbana astrological (month day year) corresponding to
  ;; fixed date.
  (let* ((new-year (solar-new-year-on-or-before
                    date 270
                    x (midnight-in-urbana x)))
         (year (round (/ (- new-year urbana-astrological-epoch)
                         mean-tropical-year)))
         (month (1+ (mod (quotient (- (solar-longitude-at-moment
                                       (midnight-in-urbana date))
                                      270)
                                   30)
                         12)))
         (day (- date -1
                 (fixed-from-urbana-astrological
                  (urbana-astrological-date month 1 year)))))
    (urbana-astrological-date month day year)))
```

For example, this function tells us that 12 November 1945 (Gregorian) = R.D. 710,347 is day 22 of month 11 of year −56 on the Urbana Astrological calendar.

The French Revolutionary Calendar

The (original) French revolutionary calendar had its new year begin on the day of the autumnal equinox in Paris using apparent (sundial) time. Thus,

```
(defun midnight-in-paris (date)
  ;; TYPE fixed-date -> moment
  ;; Moment in U.T. when it's next apparent midnight in Paris.
  (local-from-apparent-moment
   (universal-from-local (1+ date) french-time-zone)))

(defun french-new-year (f-year)
  ;; TYPE integer -> fixed-date
  ;; Fixed date of French Revolutionary new year f-year.
  (solar-event french-epoch (1- f-year) 180
               x (midnight-in-paris x)))
```

gives the fixed date of the first day of year `f-year` of the Revolution.

The Astronomical Persian Solar Calendar

The year of the astronomical Persian calendar begins on the day when the vernal equinox occurs in Teheran before noon (sundial or apparent time—the middle point of daylight) and is postponed to the next day if the equinox is after noon. Since Teheran is 3.5 hours ahead of U.T., we define

```
(defconstant tehran-time-zone
  ;; TYPE minute
  ;; The difference (in minutes) of the Teheran time zone
  ;; from Universal Time.
  210)

(defun noon-in-tehran (date)
  ;; TYPE fixed-date -> moment
  ;; Moment in U.T. when it's apparent noon in Tehran.
  (local-from-apparent-moment
   (universal-from-local (+ 1/2 date) tehran-time-zone)))
```

The fixed date of the new year is found by

```
(defun astronomical-persian-new-year (p-year)
  ;; TYPE integer -> fixed-date
  ;; Fixed date of astronomical Persian new year p-year.
  (solar-event persian-epoch (1- p-year) 0
               x (noon-in-tehran x)))
```

The Future Bahá'í Calendar

The Bahá'í year was intended by the official rules to begin at sunset following the vernal equinox. Each day begins at sunset the prior evening. The computation of the time of sunset depends on the latitude and longitude of the location on earth. Using sunset in Haifa, Israel (the exact location has yet to be determined), as the critical moment, we get:

```
(defun sunset-in-haifa (date)
  ;; TYPE fixed-date -> moment
  ;; Moment in U.T. at prior sunset in Haifa.
  (let* ((latitude 32.82)
         (longitude 35.98)
         (offset (* 4 longitude)))
    (universal-from-local
     (+ date -1 (sunset (1- date) latitude longitude))
     offset)))
```

where sunset, as given in *Calendrical Calculations*, returns the local mean time of sunset.

Thus, the epoch for the calendar is the fixed date of this event in 1844, R.D. 673,221 = 20 March 1844 (Gregorian):

```
(defconstant future-bahai-epoch
  ;; TYPE fixed-date
  ;; Fixed date of start of astronomical Bahai calendar.
  (solar-new-year-on-or-before
   (fixed-from-gregorian (gregorian-date april 1 1844))
   0 x (sunset-in-haifa x)))
```

Conversion functions for the Gregorian calendar are given in *Calendrical Calculations*.

It is now a simple matter to compute:

```
(defun future-bahai-new-year (b-year)
  ;; TYPE integer -> fixed-date
  ;; Fixed date of future Bahai calendar new year b-year
  (solar-event future-bahai-epoch (1- b-year) 0
               x (sunset-in-haifa x)))
```