Lab 3

EECS4312

September 26, 2015

Table of contents

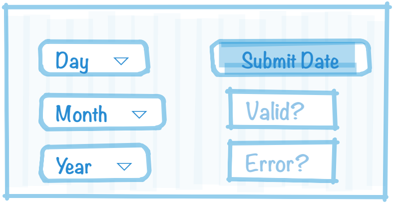
Learning Outcomes

Topics

Specify systems using function tables that can be proved to be *complete* and *disjoint*

Date Function Table 1

Figure: **User Interface to enter a Date**



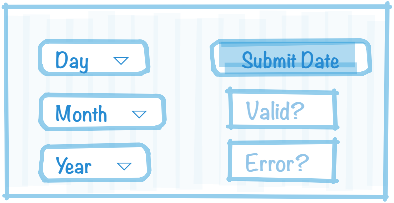
drop down box limits
day $\in$ 1..31
month $\in$ 1..12
year $\in$ 1583 ..9999

Input-Output Function Table

Given input $(day, month, year)$ at the drop boxes in the UI, provide as output whether the date is valid, and if not, what is the error.

Date Function Table 1

Figure: **User Interface to enter a Date**



drop down box limits

day ∈ 1..31

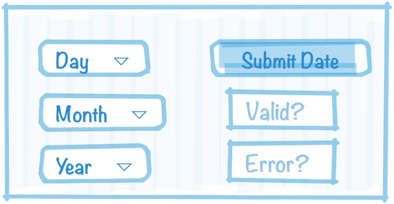
month ∈ 1..12

year ∈ 1583 ..9999

Input-Output Function Table

Given input $(day, month, year)$ at the drop boxes in the UI, provide as output whether the date is valid, and if not, what is the error.

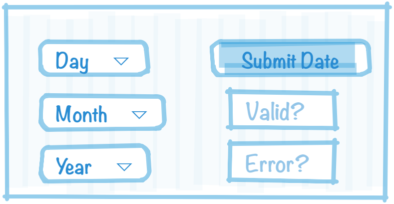
Date Function Table 2



drop down box limits

day $\in$ 1..31
month $\in$ 1..12
year $\in$ 1583 ..9999

Date Function Table 2

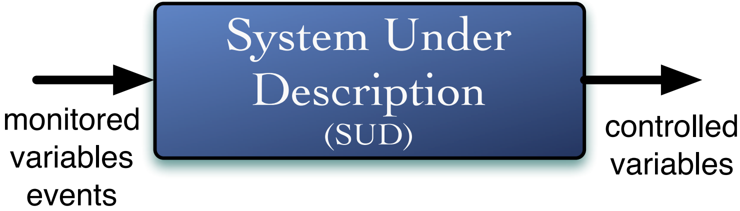


<u>drop down box limits</u>

day ∈ 1..31

month ∈ 1..12

year ∈ 1583 ..9999



Function Table to Specify Validity of a Date

<table>
<tr><th colspan="3">Input</th><th>valid?</th><th>error?</th></tr>
<tr><td colspan="3">$m \in \{1, 3, 5, 7, 8, 10, 12\}$</td><td>true</td><td>0</td></tr>
<tr><td rowspan="2">$m \in \{4, 6, 9, 11\}$</td><td colspan="2">$d \leq 30$</td><td>true</td><td>0</td></tr>
<tr><td colspan="2">$d > 30$</td><td>false</td><td>1</td></tr>
<tr><td rowspan="4">$m = 2$</td><td rowspan="2">$leapyear(y)$</td><td>$d \leq 29$</td><td>true</td><td>0</td></tr>
<tr><td>$d > 29$</td><td>false</td><td>2</td></tr>
<tr><td rowspan="2">$\neg leapyear(y)$</td><td>$d \leq 28$</td><td>true</td><td>0</td></tr>
<tr><td>$d > 28$</td><td>false</td><td>3</td></tr>
<tr><td colspan="5">Assume: $y \in 1583 \cdots 9999 \land m \in 1..12 \land d \in 1..31$</td></tr>
</table>

Date Function Table 4

<table>
<tr><th colspan="3">Input</th><th>valid?</th><th>error?</th></tr>
<tr><td colspan="3">$m \in \{1,3,5,7,8,10,12\}$</td><td>true</td><td>0</td></tr>
<tr><td rowspan="2">$m \in \{4,6,9,11\}$</td><td colspan="2">$d \leq 30$</td><td>true</td><td>0</td></tr>
<tr><td colspan="2">$d > 30$</td><td>false</td><td>1</td></tr>
<tr><td rowspan="4">$m = 2$</td><td rowspan="2">$leapyear(y)$</td><td>$d \leq 29$</td><td>true</td><td>0</td></tr>
<tr><td>$d > 29$</td><td>false</td><td>2</td></tr>
<tr><td rowspan="2">$\neg leapyear(y)$</td><td>$d \leq 28$</td><td>true</td><td>0</td></tr>
<tr><td>$d > 28$</td><td>false</td><td>3</td></tr>
<tr><td colspan="5">Assume: $y \in 1583 \cdots 9999 \wedge m \in 1..12 \wedge d \in 1..31$</td></tr>
</table>

Date Function Table

Is the function table complete and disjoint?
What is the definition of $leapyear$?

Date Function Table 4

Input			valid?	error?
$m \in \{1, 3, 5, 7, 8, 10, 12\}$			true	0
$m \in \{4, 6, 9, 11\}$	$d \leq 30$		true	0
	$d > 30$		false	1
$m = 2$	$leapyear(y)$	$d \leq 29$	true	0
		$d > 29$	false	2
	$\neg leapyear(y)$	$d \leq 28$	true	0
		$d > 28$	false	3
Assume: $y \in 1583 \cdots 9999 \wedge m \in 1..12 \wedge d \in 1..31$				

Date Function Table

Is the function table complete and disjoint?
What is the definition of $leapyear$?

Date Function Table 5: Leap Year

In the Gregorian calendar a year is a leap year if is evenly divisible by 4, and if it can be evenly divided by 100, it is **not** a leap year, unless the year is also evenly divisible by 400. This means that 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are not leap years.

How to formalize?

leap_year(y: $\mathbb{N}$): $\mathbb{B}$
require $y \geq 1583$
ensure

$$Result = mod(y,4)=0 \wedge mod(y,400) \notin \{100,200,300\}$$

Date Function Table 5: Leap Year

In the Gregorian calendar a year is a leap year if is evenly divisible by 4, and if it can be evenly divided by 100, it is **not** a leap year, unless the year is also evenly divisible by 400. This means that 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are not leap years.

How to formalize?

leap_year(y: $\mathbb{N}$): $\mathbb{B}$

require $y \geq 1583$

ensure

$$Result \;=\; mod(y,4) = 0 \;\wedge\; mod(y,400) \notin \{100, 200, 300\}$$

Date Function Table 5: Leap Year

In the Gregorian calendar a year is a leap year if is evenly divisible by 4, and if it can be evenly divided by 100, it is **not** a leap year, unless the year is also evenly divisible by 400. This means that 2000 and 2400 are leap years, while 1800, 1900, 2100, 2200, 2300 and 2500 are not leap years.

How to formalize?

leap_year(*y*: $\mathbb{N}$): $\mathbb{B}$
require $y \geq 1583$
ensure

$$Result \quad = \quad mod(y, 4) = 0 \ \wedge \ mod(y, 400) \notin \{100, 200, 300\}$$

Date Function Table 6: Leap Year

leap_year(y: $\mathbb{N}$): $\mathbb{B}$
  require $y \geq 1583$
  ensure $mod(y, 4) = 0 \wedge mod(y, 400) \notin \{100, 200, 300\}$

```
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}
% Note that a set is just a predicate
set_conjecture1: CONJECTURE
      LEAP(100) AND NOT LEAP(150)
  AND member(100,LEAP)
```

Date Function Table 6: Leap Year

leap_year(y: $\mathbb{N}$): $\mathbb{B}$
require $y \geq 1583$
ensure $mod(y, 4) = 0 \wedge mod(y, 400) \notin \{100, 200, 300\}$

```
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}
% Note that a set is just a predicate
set_conjecture1: CONJECTURE
      LEAP(100) AND NOT LEAP(150)
  AND member(100,LEAP)
```

Date Function Table: Leap Year


```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}

% From prelude for mod:
%    mod(7,2) = 1 = (remainder when 7/2)
% (consistent with Ada definition):
% M-x view-prelude_theory
% i: VAR int
% j: VAR nonzero_integer
% mod(i,j): {k| abs(k) < abs(j)} = i-j*floor(i/j)

% definition of leap year
leapyr(y: YEAR): bool =
      mod(y,4) = 0
  AND NOT member(mod(y,400),LEAP)
```

Date Function Table: Leap Year


```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}

% From prelude for mod:
%    mod(7,2) = 1 = (remainder when 7/2)
% (consistent with Ada definition):
% M-x view-prelude_theory
% i: VAR int
% j: VAR nonzero_integer
% mod(i,j): {k| abs(k) < abs(j)} = i-j*floor(i/j)

% definition of leap year
leapyr(y: YEAR): bool =
      mod(y,4) = 0
  AND NOT member(mod(y,400),LEAP)
```

Date Function Table: Leap Year


```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}

% From prelude for mod:
%    mod(7,2) = 1 = (remainder when 7/2)
% (consistent with Ada definition):
% M-x view-prelude_theory
% i: VAR int
% j: VAR nonzero_integer
% mod(i,j): {k| abs(k) < abs(j)} = i-j*floor(i/j)

% definition of leap year
leapyr(y: YEAR): bool =
      mod(y,4) = 0
  AND NOT member(mod(y,400),LEAP)
```

Date Function Table: Leap Year

```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}

% definition of leap year
leapyr(y: YEAR): bool =
  mod(y,4) = 0 AND NOT member(mod(y,400),LEAP)

% TCC unprovable
conj0: CONJECTURE
   NOT leapyr(1582)

% Subtype TCC generated
% expected type  YEAR
% unfinished
%conj0_TCC1: OBLIGATION 1583 <= 1582
%     AND 1582 <= 9999;
```

Date Function Table: Leap Year


```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}
% definition of leap year
leapyr(y: YEAR): bool =
  mod(y,4) = 0 AND NOT member(mod(y,400),LEAP)

% TCC unprovable
conj0: CONJECTURE
   NOT leapyr(1582)

% Subtype TCC generated
% expected type  YEAR
% unfinished
%conj0_TCC1: OBLIGATION 1583 <= 1582
%    AND 1582 <= 9999;
```

Date Function Table: Leap Year


```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                    OR x = 300}

% definition of leap year
leapyr(y: YEAR): bool =
  mod(y,4) = 0
  AND NOT member(mod(y,400),LEAP)

conj1: CONJECTURE
  NOT leapyr(1583)

conj3: CONJECTURE
  NOT leapyr(1900)

conj4: CONJECTURE
  leapyr(2000)
```

Date Function Table: Leap Year


```
YEAR : TYPE = subrange(1583,9999)
LEAP: set[nat] = {x: nat | x= 100  OR x = 200
                                   OR x = 300}

% definition of leap year
leapyr(y: YEAR): bool =
  mod(y,4) = 0
  AND NOT member(mod(y,400),LEAP)

conj1: CONJECTURE
  NOT leapyr(1583)

conj3: CONJECTURE
  NOT leapyr(1900)

conj4: CONJECTURE
  leapyr(2000)
```

Date Function Table: PVS version

Input			valid?	error?
$m \in \{1,3,5,7,8,10,12\}$			true	0
$m \in \{4,6,9,11\}$	$d \leq 30$		true	0
	$d > 30$		false	1
$m = 2$	$leapyear(y)$	$d \leq 29$	true	0
		$d > 29$	false	2
	$\neg leapyear(y)$	$d \leq 28$	true	0
		$d > 28$	false	3
Assume: $y \in 1583 \cdots 9999 \wedge m \in 1..12 \wedge d \in 1..31$				

```
YEAR : TYPE = subrange(1583,9999)
MONTH: TYPE = subrange(1,12)
DAY :  TYPE = subrange(1,31)

LEAP: set[nat] = {x: nat | x= 100 OR x = 200 OR x = 300}
MONTH31: set[MONTH] = {x: MONTH | x= 1 OR x = 3 OR x = 5
                        OR x=7 OR x=8 OR x=10 OR x=12}
MONTH30: set[MONTH] = {x: MONTH | x= 4 OR x = 6 OR x = 9
                        OR x = 11}

leapyr(y: YEAR): bool =
mod(y,4) = 0
AND NOT member(mod(y,400),LEAP)

%input variables (undetermined constants)
y: YEAR
m: MONTH
d: DAY

% output variables
valid: bool
err: nat

date_valid: bool =
COND
   MONTH31(m)
-> valid = True AND err = 0,
   MONTH30(m) AND d <= 30
-> ....???
ENDCOND

date_validity_check1: CONJECTURE
  date_valid => NOT (valid = True AND err > 0)

date_validity_check2: CONJECTURE
  date_valid =>  (0 <= err AND err <= 3)
```

Date Function Table
Complete it and
prove all the TCCs
and conjectures.

The TTCs check for completeness
and disjointness

Validate Function Table Specification

Date Function Table: PVS version


```
YEAR : TYPE = subrange(1583,9999)
MONTH: TYPE = subrange(1,12)
DAY :  TYPE = subrange(1,31)

LEAP: set[nat] = {x: nat | x= 100 OR x = 200 OR x = 300}
MONTH31: set[MONTH] = {x: MONTH | x= 1 OR x = 3 OR x = 5
                       OR x=7 OR x=8 OR x=10 OR x=12}
MONTH30: set[MONTH] = {x: MONTH | x= 4 OR x = 6 OR x = 9
                       OR x = 11}

leapyr(y: YEAR): bool =
mod(y,4) = 0
AND NOT member(mod(y,400),LEAP)

%input variables (undetermined constants)
y: YEAR
m: MONTH
d: DAY

% output variables
valid: bool
err: nat

date_valid: bool =
COND
   MONTH31(m)
-> valid = True AND err = 0,
   MONTH30(m) AND d <= 30
-> ....???
ENDCOND

date_validity_check1: CONJECTURE
  date_valid => NOT (valid = True AND err > 0)

date_validity_check2: CONJECTURE
  date_valid =>  (0 <= err AND err <= 3)
```

Date Function Table
Complete it and
prove all the TCCs
and conjectures.

The TTCs check for completeness
and disjointness

Validate Function Table Specification

Completeness and Disjointness

Input conditions		*Output* r
$C_1(x)$	$C_{11}(x)$	$R_1(x, r)$
	$C_{12}(x)$	$R_2(x, r)$
$C_3(x)$		$R_3(x, r)$
assume: A		

$$P_1 \triangleq C_1(x) \wedge C_{11}(x)$$
$$P_2 \triangleq C_1(x) \wedge C_{12}(x)$$
$$P_3 \triangleq C_3(x)$$
$$Q_i \triangleq R_i(x, r) \text{ for } i \in 1 \,..\, 3$$

(a) **Meaning of Table** : $A \Rightarrow (\forall i \in 1 \,..\, 3 : P_i \Rightarrow Q_i)$

(b) **Completeness** : $A \Rightarrow (\exists i \in 1 \,..\, 3 : P_i)$

(c) **Disjointness** : $A \Rightarrow (\forall i, j \in 1 \,..\, 3 \;:\; i \neq j : \neg(P_i \wedge P_j))$

(d) **Well-definedness** :

$$D(A) \;\wedge\; (A \wedge (\forall i \in 1 \,..\, 3 : \mathcal{D}(P_i) \wedge (P_i \Rightarrow \mathcal{D}(Q_i)))$$

Figure: Completeness, Disjointness and Well-definedness of tabular expressions

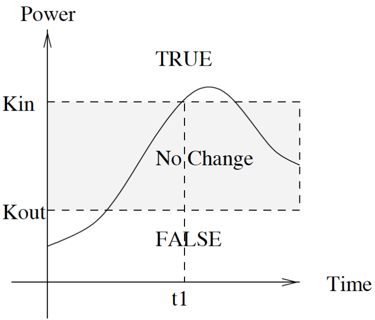
Date Function Table: PVS version

Status of Date Theory

You should see the following, when done:

```
Proof summary for theory date
    set_conjecture1.......................proved - complete   [shostak](0.01 s)
    conj1.................................proved - complete   [shostak](0.05 s)
    conj3.................................proved - complete   [shostak](0.07 s)
    conj4.................................proved - complete   [shostak](0.07 s)
    y_TCC1................................proved - complete   [shostak](0.01 s)
    m_TCC1................................proved - complete   [shostak](0.01 s)
    d_TCC1................................proved - complete   [shostak](0.00 s)
    date_valid_TCC1.......................proved - complete   [shostak](0.66 s)
    date_valid_TCC2.......................proved - complete   [shostak](0.19 s)
    date_validity_check1..................proved - complete   [shostak](0.10 s)
    date_validity_check2..................proved - complete   [shostak](0.14 s)
    Theory totals: 11 formulas, 11 attempted, 11 succeeded (1.32 s)
```

Specification of Power Conditioning in a Nuclear Reactor

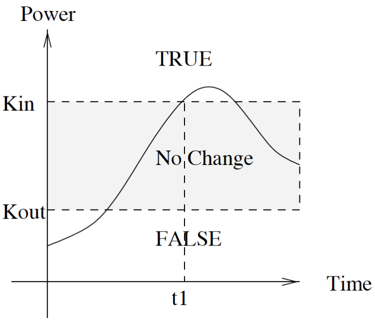


The figure to the left illustrates a power conditioning function that is used in an industrial control system. When the Power level drops below `Kout`, a sensor becomes unreliable so it is "conditioned out" by setting `PwrCond` to FALSE. When the power exceeds `Kin`, the sensor is "conditioned in" and is used to evaluate the system. While `Power` is between `Kout` and `Kin`, the value of `PwrCond` is left unchanged by setting it to its previous value, `Prev`.

Required Specification

Given that different sensors might have different values for *Kin* and *Kout*, a general power condition function is proposed.

Specification of Power Conditioning in a Nuclear Reactor



The figure to the left illustrates a power conditioning function that is used in an industrial control system. When the Power level drops below `Kout`, a sensor becomes unreliable so it is "conditioned out" by setting `PwrCond` to FALSE. When the power exceeds `Kin`, the sensor is "conditioned in" and is used to evaluate the system. While `Power` is between `Kout` and `Kin`, the value of `PwrCond` is left unchanged by setting it to its previous value, `Prev`.

Required Specification

Given that different sensors might have different values for *Kin* and *Kout*, a general power condition function is proposed.

PVS Specification of Power Conditioning in a Nuclear Reactor


```
power: THEORY
BEGIN
  x: VAR real

  % Function Tables Using TABLE
  g(x): real = TABLE
      %-------------%
      |[ x<0 | x>=0 ]|
      %-------------%
      |  x   | 2*x  ||
      %-------------%
  ENDTABLE

  h(x): real = TABLE
      %-------------%
      |[ x< 0 | x>=0 ]|
      %-------------%
      |  x   | x + x ||
      %-------------%
  ENDTABLE

  same: THEOREM FORALL (x:real): g(x)=h(x)

  % Note how we deal with the previous instant
  PwrCond(Prev:bool, Power, Kin, Kout:posreal):bool = TABLE
   %---------------------------------------------%
   |[Power<=Kout|Power>Kout & Power<Kin|Power>=Kin]|
   %---------------------------------------------%
   | TRUE       | Prev                 | FALSE    ||
   %---------------------------------------------%
   ENDTABLE

END power
```

PVS files provided with top.pvs


```
% Exercises for Lab3
% proveit --importchain --clean top.pvs
top : THEORY
BEGIN
   IMPORTING date
   IMPORTING power
END top
```

PVS files

Work through the PVS files, following the instructions in each.

Result of running proveit on top.pvs


```
Proof summary for theory top
   Theory totals: 0 formulas, 0 attempted, 0 succeeded (0.00 s)

Proof summary for theory date
   set_conjecture1......................proved - complete   [shostak](0.02 s)
   conj1................................proved - complete   [shostak](0.06 s)
   conj3................................proved - complete   [shostak](0.06 s)
   conj4................................proved - complete   [shostak](0.07 s)
   y_TCC1...............................proved - complete   [shostak](0.00 s)
   m_TCC1...............................proved - complete   [shostak](0.00 s)
   d_TCC1...............................proved - complete   [shostak](0.01 s)
   date_valid_TCC1......................proved - complete   [shostak](0.66 s)
   date_valid_TCC2......................proved - complete   [shostak](0.17 s)
   date_validity_check1.................proved - complete   [shostak](0.10 s)
   date_validity_check2.................proved - complete   [shostak](0.11 s)
   Theory totals: 11 formulas, 11 attempted, 11 succeeded (1.27 s)

Proof summary for theory power
   g_TCC1...............................proved - complete   [shostak](0.00 s)
   g_TCC2...............................proved - complete   [shostak](0.00 s)
   same.................................proved - complete   [shostak](0.00 s)
   PwrCond_TCC1.........................unfinished         [shostak](0.05 s)
   PwrCond_TCC2.........................proved - complete   [shostak](0.01 s)
   Theory totals: 5 formulas, 5 attempted, 4 succeeded (0.07 s)

Grand Totals: 16 proofs, 16 attempted, 15 succeeded (1.34 s)
```

Submit your work 1

- Rename the directory with your PVS files to `4312-lab3`, and then run the following command in the directory:
- `proveit --importchain --clean top.pvs`
- Remove all the pdf files from the directory. You must only submit the pvs files ***.pvs** and ***.prf**. Delete everything else.

Submit your work 2

- Now submit your `4312-lab3` directory:
  `> submit 4312 lab3 4312-lab3`
- You will get confirmation of your submission.
- To obtain a grade on your quiz, you must complete and submit this Lab.