Flight Schedule Database Example

by

Ricky W. Butler
NASA, Langley Research Center

May 19, 1995

Flight Schedule Example

Requirements for an Airport Flight Schedule Database

- The flight schedule database shall store the scheduling information associated with all departing and arriving flights. In particular the database shall contain:
  - departure time and gate number
  - arrival time and gate number
  - route (i.e. navigation way points)

  for each arriving and departing flight.
- There shall be a way to retrieve the scheduling information given a flight number.
- It shall be possible to add and delete flights from the database.

Formal Requirements Specification

- **How do we represent the flight schedule database mathematically?**

  1. **a set of ordered pairs of flight numbers and schedules. Adding and deleting entries via set addition and deletion**

  2. **function whose domain is all possible flight numbers and range is all possible schedules. Adding and deleting entries via modification of function values.**

  3. **function whose domain is only flight numbers currently in database and range is the schedules. Adding and deleting entries via modification of the function domain and values.**

Note: The choice between these is strongly influenced by the verification system used.

Getting Started

Let's start with approach 2:

> **function whose domain is all possible flight numbers and range is all possible schedules. Adding and deleting entries via modification of function values.**

In traditional mathematical notation, we would write:

$$\textbf{Let } N = \textbf{set of flight numbers}$$

$$S = \textbf{set of schedules}$$

$$D : N \longrightarrow S$$

where D represents the database and S represents all of the schedule information.

Note that the details have been *abstracted away*. This is one of the most important steps in producing a good formal specification.

Specifying the Flight Schedule Database

$$D : N \longrightarrow S$$

How do we indicate that we do not have a flight schedule for all possible flight numbers?

We declare a constant of type S, say "u_o", that indicates that there is no flight scheduled for this flight number.

Now can define an empty database. In traditional notation, we would write:

$$empty\_database : N \longrightarrow S$$

$$empty\_database(flt) \equiv u_o$$

$$\forall \; flt \in N$$

Accessing an Entry

Let $N =$ **set of flight numbers**

$S =$ **set of schedules**

$D =$ **set of functions** $: N \longrightarrow S$

$\forall d \in D$ **and** $flt \in N$.

$$find\_schedule : D \times N \longrightarrow S$$

$$find\_schedule(d, flt) = d(flt)$$

Note that $find\_schedule$ **is a higher-order function since its first argument is a function.**

Specifying Adding/Deleting an Entry

Let $N =$ **set of flight numbers**

$S =$ **set of schedules**

$D : N \longrightarrow S$

$u_o \in S$

$D =$ **set of functions** $: N \longrightarrow S$

$\forall d \in D,\ \ \forall flt \in N,\ \ \forall sched \in S$

$$add\_flight : D \times N \times S \longrightarrow D$$

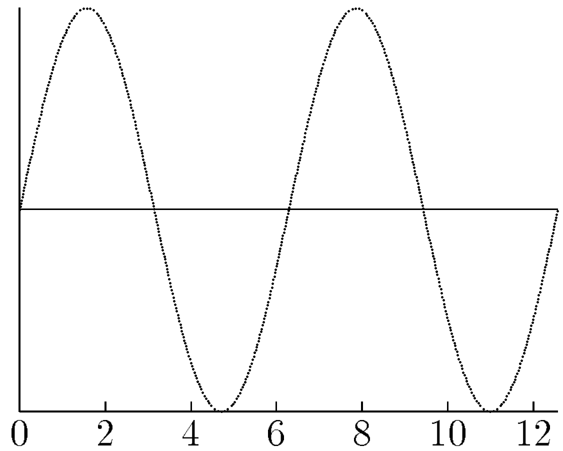
$$add\_flight(d, flt, sched)(x) = \begin{cases} d(x) & \textbf{if } x \neq flt \\ sched & \textbf{if } x = flt \end{cases}$$

$$delete\_flight : D \times N \longrightarrow D$$

$$delete\_flight(d, flt)(x) = \begin{cases} d(x) & \textbf{if } x \neq flt \\ u_o & \textbf{if } x = flt \end{cases}$$

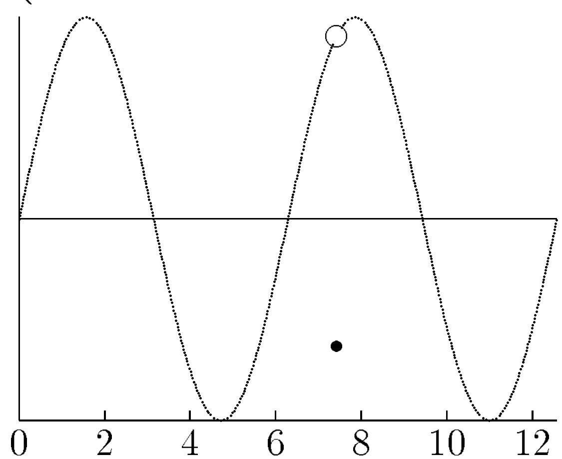
The WITH Notation

sin(x):



$$\textbf{sin(x) WITH [7.4 := -0.60]} = \begin{cases} \textbf{-0.60} & \textbf{if } x = \textbf{7.4} \\ sin(x) & \textbf{otherwise} \end{cases}$$

sin(x) WITH [7.4 := -0.60]



Complete Spec (Omitting Function Signatures)

Let $N =$ **set of flight numbers**
$S =$ **set of schedules**
$D =$ **set of functions** $: N \longrightarrow S$
$\forall d \in D, \;\; \forall flt \in N, \;\; \forall sched \in S$

$$find\_schedule(d, flt) = d(flt)$$

$$add\_flight(d, flt, sched)(x) = d \textbf{ WITH } [flt := sched]$$

$$delete\_flight(d, flt)(x) = d \textbf{ WITH } [flt := u_o]$$

Can test spec with some putative theorems:

LEMMA (*putative 1*) : $find\_schedule(add\_flight(d, flt, sched), flt) = sched$
LEMMA (*putative 2*) : $delete\_flight(add\_flight(d, flt, sched), flt) = d$

Attempted Verification Of Putative 2 Reveals a Problem

LEMMA ($\textit{putative 2}$)**:** $delete\_flight(add\_flight(d, flt, sched), flt) = d$
Proof:

$$delete\_flight(add\_flight(d, flt, sched), flt) =$$

$$delete\_flight(d \textbf{ WITH } [flt := sched]) =$$

$$d \textbf{ WITH } [flt := sched] \textbf{ WITH } [flt := u_o] =$$

$$d \textbf{ WITH } [flt := u_o] = ??$$

But there is no way to reach d, because

$$d \textbf{ WITH } [flt := u_o] \neq d$$

unless $d(flt) = u_o$.

This is only true if the flt is currently not scheduled in the flight database.

Verification Reveals Oversight

- **We realize that we only want to add a flight with flight number flt, if one is not already in the database.**
- **If flt is already in the database, we probably need the capability to change it.**

Thus, we modify add_flight and create a new function $change\_flight$:

Verification Reveals Oversight (Cont.)

Let $N =$ **set of flight numbers**

$S =$ **set of schedules**

$D =$ **set of functions** $: N \longrightarrow S$

$\forall d \in D, \ \ \forall flt \in N, \ \ \forall sched \in S$

$$scheduled?(d, flt) : boolean = d(flt) \neq u_o$$

$$add\_flight(d, flt, sched) =$$

$$\textbf{IF } scheduled?(d, flt) \textbf{ THEN } d$$

$$\textbf{ELSE } d \textbf{ WITH } [flt := sched] \textbf{ ENDIF}$$

$$change\_flight(d, flt, sched) =$$

$$\textbf{IF } scheduled?(d, flt) \textbf{ THEN } d \textbf{ WITH } [flt := sched]$$

$$\textbf{ELSE } d \textbf{ ENDIF}$$

Putative 2 Proof After Correction

LEMMA (*putative 2*): **NOT** $scheduled?(d, flt) \supset$

$$delete\_flight(add\_flight(d, flt, sched), flt) = d$$

Proof:

$$delete\_flight(add\_flight(d, flt, sched), flt)$$

$$= delete\_flight(\ \mathbf{IF}\ scheduled?(d, flt)\ \mathbf{THEN}\ d$$
$$\mathbf{ELSE}\ d\ \mathbf{WITH}\ [flt := sched]\ \mathbf{ENDIF}\ )$$

$$= delete\_flight(d\ \mathbf{WITH}\ [flt := sched])$$

$$= d\ \mathbf{WITH}\ [flt := sched]\ \mathbf{WITH}\ [flt := u_o]$$

$$= d\ \mathbf{WITH}\ [flt := u_o]$$

$$= d \qquad \textbf{(because NOT } scheduled?(d, flt) \supset d(flt) = u_o\ \textbf{)}$$

A Minor Problem

To check our new function `schedule?` we postulate the following putative theorem:

$$\textbf{SchedAdd: LEMMA } scheduled?(add\_flight(d, flt, sched), flt)$$

Proof:

$$scheduled?(add\_flight(d, flt, sched)) =$$

$$scheduled?(\textbf{ IF } scheduled?(d, flt) \textbf{ THEN } d$$

$$\textbf{ELSE } d \textbf{ WITH } [flt := sched] \textbf{ ENDIF } =$$

$$\textbf{IF } d(flt) \neq u_o \textbf{ THEN } d(flt) \neq u_o$$

$$\textbf{ELSE } d \textbf{ WITH } [flt := sched](flt) \neq u_o \textbf{ ENDIF } =$$

$$d \textbf{ WITH } [flt := sched](flt) \neq u_o$$

$$sched \neq u_o$$

which is not provable because nothing prevents $sched = u_o$**.**

A Minor Problem Repaired

We then realize that our specification does not rule out the possibility of assigning a "u_o" schedule to a real flight

$$
\begin{aligned}
&\textbf{Let } N = \textbf{set of flight numbers}\\
&\quad S = \textbf{set of schedules}\\
&\quad S^* = \textbf{set of schedules not including } u_o\\
&\quad D = \textbf{set of functions} : N \longrightarrow S\\
&\quad \forall d \in D,\ \ \forall flt \in N,\ \ \forall sched \in S^*\\
&find\_schedule : D \times N \longrightarrow S\\
&add\_flight : D \times N \times S^* \longrightarrow D\\
&change\_flight : D \times N \times S^* \longrightarrow D\\
&delete\_flight : D \times N \longrightarrow D
\end{aligned}
$$

This type of trivial problem is usually not manifested until when one attempts a mechanical (i.e. level 3) verification.

Another Example of a Putative Theorem

$$
\begin{array}{ll}
 & (\forall i : flt_i \neq flt) \wedge \\
 & \\
 & find\_schedule(d_0, flt) = sched \wedge \\
 & d_1 = add\_flight(d_0, flt_1, sched_1) \wedge \\
 & d_2 = add\_flight(d_1, flt_2, sched_2) \wedge \\
 & \quad . \qquad\qquad\qquad\qquad\qquad . \\
 & \quad . \qquad\qquad\qquad\qquad\qquad . \\
 & \quad . \qquad\qquad\qquad\qquad\qquad . \\
 & d_n = add\_flight(d_{n-1}, flt_n, sched_n) \\
\supset & \\
 & find\_schedule(d_n, flt) = sched
\end{array}
$$

- **Formal methods can establish that even in the presence of an** *arbitrary* **number of operations a property holds.**
- **Testing can never establish this.**

Some Observations

- Our specification is abstract. The functions are defined over infinite domains.
- As one translates the requirements into mathematics, many things that are usually left out of English specifications are explicitly enumerated.
- The formal process exposes ambiguities and deficiencies in the requirements.
- Putative theorem proving and scrutiny reveals deficiencies in the formal specification.

PVS Spec

```
flight_sched3: THEORY
BEGIN

  N : TYPE                          % flight numbers
  S : TYPE                          % schedules
  D : TYPE = [N -> S]               % flight database

  u0: S                             % unscheduled

  S_good  : TYPE = {sched: S | sched /= u0}

  flt   : VAR N
  d     : VAR D
  sched : VAR S_good

  emptydb(flt): S = u0

  find_schedule(d, flt): S = d(flt)

  scheduled?(d,flt): boolean = d(flt) /= u0
```

```
  add_flight(d, flt, sched): D =
       IF scheduled?(d,flt) THEN d
       ELSE d WITH [flt := sched] ENDIF

  change_flight(d, flt, sched): D =
       IF scheduled?(d,flt) THEN d WITH [flt := sched]
       ELSE d ENDIF

  delete_flight(d, flt): D = d WITH [flt := u0]


  putative2 : LEMMA NOT scheduled?(d,flt) IMPLIES
                        delete_flight(add_flight(d,flt,sched),flt) = d

  SchedAdd  : LEMMA scheduled?(add_flight(d,flt,sched),flt)


END flight_sched3
```

Introduction to a PVS Proof

- **Illustrative proof**
- **The single command `GRIND` proves it automatically**

```
putative2 :

  |-------
{1}   (FORALL (d: D, flt: N, sched: S_good):
         NOT scheduled?(d, flt)
           IMPLIES delete_flight(add_flight(d, flt, sched), flt) = d)

Rule? (SKOSIMP*)
Repeatedly Skolemizing and flattening,
this simplifies to:
putative2 :

  |-------
{1}   scheduled?(d!1, flt!1)
{2}   delete_flight(add_flight(d!1, flt!1, sched!1), flt!1) = d!1

Rule? (EXPAND "add_flight" )
Expanding the definition of add_flight,
```

```
this simplifies to:
putative2 :

  |-------
[1]   scheduled?(d!1, flt!1)
{2}   delete_flight(IF scheduled?(d!1, flt!1) THEN d!1
                 ELSE d!1 WITH [flt!1 := sched!1]
                 ENDIF,
                 flt!1)
        = d!1

Rule? (LIFT-IF )
Lifting IF-conditions to the top level,
this simplifies to:
putative2 :

  |-------
[1]   scheduled?(d!1, flt!1)
{2}   IF scheduled?(d!1, flt!1) THEN delete_flight(d!1, flt!1) = d!1
      ELSE delete_flight(d!1 WITH [flt!1 := sched!1], flt!1) = d!1
      ENDIF

Rule? (ASSERT)
Simplifying, rewriting, and recording with decision procedures,
```

```
this simplifies to:
putative2 :

  |-------
[1]   scheduled?(d!1, flt!1)
{2}   delete_flight(d!1 WITH [flt!1 := sched!1], flt!1) = d!1

Rule? (EXPAND "delete_flight" )
Expanding the definition of delete_flight,
this simplifies to:
putative2 :

  |-------
[1]   scheduled?(d!1, flt!1)
{2}   d!1 WITH [flt!1 := sched!1] WITH [flt!1 := u0] = d!1

Rule? (EXPAND "scheduled?" )
Expanding the definition of scheduled?,
this simplifies to:
putative2 :

  |-------
{1}   d!1(flt!1) /= u0
[2]   d!1 WITH [flt!1 := sched!1] WITH [flt!1 := u0] = d!1
```

```
Rule? (assert)
Simplifying, rewriting, and recording with decision procedures,
Q.E.D.


Run time  = 1.16 secs.
Real time = 61.49 secs.

Wrote proof file /airlab/home/rwb/fm/wkshp/pvs/flight_sched3.prf
NIL
>
```

New Requirement

"Two flights are not to be scheduled at the same gate at the same time!"

Introduce refinement of schedule:

```
N  : TYPE
Date_and_time: TYPE
Gate_nums: TYPE
A_or_D: TYPE = (arriving,departing)
Way: TYPE

S: TYPE = [# % RECORD
          departure_tm: Date_and_time,
          arrival_tm: Date_and_time,
          dep_gate: Gate_nums,
          arr_gate: Gate_nums,
          arr_or_dep: A_or_D,
          nav_route: Way
          #] % END RECORD
```

Simplified Problem

Often it is useful to solve a simplified problem before you tackle the big problem. So let's only work with departing flights:

```
N  : TYPE
Date_and_time: TYPE
Gate_nums: TYPE

S: TYPE = [# % RECORD
          departure_tm: Date_and_time,
          dep_gate: Gate_nums,
          #] % END RECORD
```

The requirement states that "two flights are not to be scheduled at the same gate at the same time":

```
same_time(sched1, sched2): boolean
```

New Requirement Continued

We also need to introduce concept of "scheduled at the same gate at the same time":

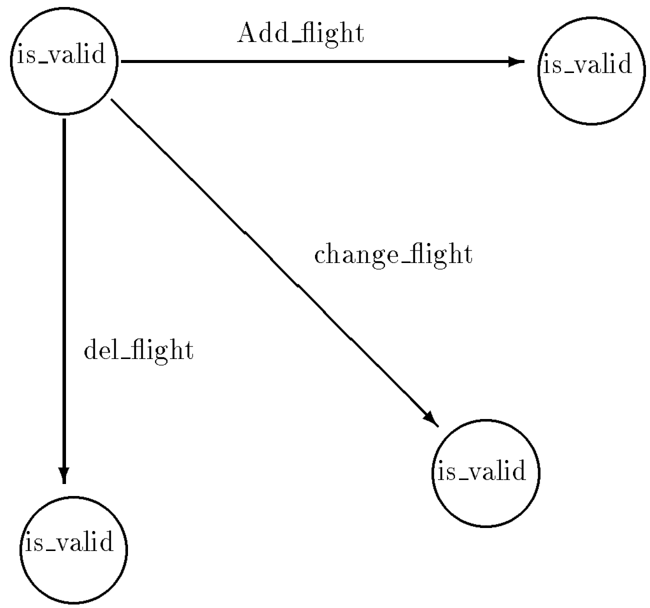
```
overlapped(sched1,sched2): boolean = dep_gate(sched1) = dep_gate(sched2)
                                     AND same_time(sched1,sched2)
```

We would like to establish that the operations on the database will never result in an `overlapped` **situation.**

In other words, we want to establish an *invariant*:

```
is_valid(d: D): boolean = (FORALL (flt1,flt2: N): flt1 /= flt2 AND
           scheduled?(d,flt1) AND scheduled?(d,flt2) IMPLIES
                NOT overlapped(find_schedule(d,flt1), find_schedule(d,flt2)))
```

Database System as a State Machine



Need to establish that all of the "operations" maintain the invariant. For example,

```
add_flight_is_valid: LEMMA
      (FORALL (d: D, flt: N, sched: S_good):
            is_valid(d) IMPLIES is_valid(add_flight(d,flt,sched)));
```

Of course, `add_flight` **must be modified to insure that this is true:**

Add_flight Modified To Maintain Invariant

```
gate_in_use_at_time(d,sched): boolean =
       (EXISTS flt: scheduled?(d,flt) AND overlapped(sched,d(flt)))

add_flight(d, flt, sched): D =
     IF scheduled?(d,flt) OR gate_in_use_at_time(d,sched) THEN d
     ELSE d WITH [flt := sched]
     ENDIF
```

Thus, we have modeled the database as a finite state machine and the functions `add_flight`, `change_flight`, **and** `delete_flight` **are operations on the state machine.**

State Machines and PVS Type System

By creating a predicate subtype of the type D:

```
Valid_db:    TYPE = {d: D | is_valid(d)}
```

and modifying the signatures of `add_flight`, `change_flight`, **and** `delete_flight`, **e.g.**

```
add_flight(vd: Valid_db, flt, sched): Valid_db =
     IF scheduled?(vd,flt) OR gate_in_use_at_time(vd,sched) THEN vd
     ELSE vd WITH [flt := sched]
     ENDIF
```

PVS will automatically generate the "invariant" lemmas that must proved (called TCC's).

- **is just a particular case of the more general TCC mechanism**
- **illustrates how a mechanized specification language can provide much stronger typechecking than traditional programming languages**

Conclusions

- **With formal methods a clear, unambiguous, abstract specification can be constructed.**
- **Mechanized formal methods allows you can CALCULATE (prove) whether the specification has certain properties.**
- **These calculations can be done early in the lifecycle on abstract descriptions.**
- **And they can cover ALL the case,.**