Development of NASA Langley's PVS Libraries

Ricky W. Butler

r.w.butler@larc.nasa.gov

http://shemesh.larc.nasa.gov/fm/

Oct 27, 2003

Practical/Universal Digital Mathematics Libraries: A Big Challenge

My talk:

List of NASA LaRC PVS Libraries

algebra	groups, monoids, rings, etc.	details
analysis	real analysis, limits, continuity, derivatives	details
array	min, max, majority, sort, concatenation over arrays + permutations	details
bags	bags, finite bags, inductions, conversions to sets	details
calculus	axiomatic version of calculus	details
digraphs	directed graphs: circuits, maximal subtrees, paths, dags	details
div	integer division: Ada and number theory definitions	details
fixedpoints	fixed-points of `[set[T] -> set[T]]`, mu-calculus, dual(H)	details
graphs	graph theory: connectedness, walks, trees, Menger's Theorem	details
mod	modulo arithmetic: `mod(i,j): {k| abs(k) < abs(j)} = i-j*floor(i/j)`	details
nat_funs	special functions over nats: min, max, mod, div	details
number_theory	fundamental theorem arithmetic, primes infinite cardinality	details
reals	summations, sup, inf, sqrt over the reals, abs lemmas	details
series	power series, comparison test, ratio test, Taylor's theorem	details
trig	trigonometry: definitions, identities, approximations	details
powersets	powersets: bijections, cardinality	details
vectors	2D,3D and N-dimen vectors, intersection, pt. closest approach	details

STRATEGIES

Name	Documentation	Concept
Field	README-field	remove divisions from a formula
Manip	manip.ps	algebraic manipulation of formulas
Test Probs	test-prob-doc.txt	theorem prover benchmark problems

http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html

What are the Main Challenges?

Many Unguided Tradeoffs:

- **Foundational versus Efficient**
- **Mathematical generality and elegance versus simplicity and ease of use**
- **Completeness of theories verses how easy to locate things**
- **Multiple definitions – which one?**
- **How many different styles of proof do you support?**
- **Which rewrites do you make automatic?**
- **Layout and style**
- **Maintenance in the presence of an evolving PVS**

Not a Whole Lot of Glory in Developing Libraries

Hey, Michael, “I just completed the proof of the Fundamental Theorem of Calculus in PVS?”

“I thought that was completed over 300 years ago?”

“Yes, but this is a rigorous proof!”

“Didn’t Cauchy make analysis rigorous in the 19th century?”

“Yes, but this time it is in PVS!!”

“Yeah, that’s really nice.”

Not a Whole Lot of Glory in Developing Them (cont.)

In the NASA library there is one remaining unproved result before the Fundamental Theorem will be complete:

```
integral_split: LEMMA a < b AND b < c AND integrable?(a,b,f) AND
                                          integrable?(b,c,f)
                          IMPLIES integrable?(a,c,f) AND
                integral(a,b,f) + integral(b,c,f) = integral(a,c,f)
```

Any Volunteers?

- **I estimate that this lemma will take about one month.**
- **Since this kind of tedious work is not really publishable, hard to find people to do it.**
- **Much easier to get the "meta stuff" published: e.g., translating from one formalism to another**

Foundational versus Efficient

In the trig library you will find about 20+ axioms, such as:

```
sin,cos      : [real -> real]

sin_range_ax: AXIOM -1 <= sin(a) AND sin(a) <= 1
cos_range_ax: AXIOM -1 <= cos(a) AND cos(a) <= 1

sin2_cos2    : AXIOM sq(sin(a)) + sq(cos(a)) = 1

cos_0        : AXIOM cos(0) = 1
```

From these over two hundred theorems are proved.

Now in the series library several of these axioms are proved.

- **If we finally got around to proving them all**
- **Do you want to import**
  - **series library AND**
  - **analysis library AND**
  - **real library**

every time you need to use `cos`**?**

Foundational versus Efficient (cont.)

Should we have different libraries, e.g. calculus and analysis?

- **How do we keep them consistent?**
- **What goes in each one?**
- **How do we organize them?**

At NASA Langley we set up a team to look over a library:

- **We debate definitions, structure, etc.**
- **One of the most important things: naming of lemmas**
  - **nice conventions, e.g.** `sqrt_0`, `cos_0`
  - **Difficult to remember when you have different ones: e.g.** `_times` `_mult`, **or** `_plus` `_add`, **or** `_0` `_zero`**...**
  - **Should we have a glossary of common theorem name parts:** `distr` **for distributive,** `assoc` **for associative,** `eq_args` **for** $f(a, a) = ...$**,** `_eq_0` **for** $f(...) = 0 \supset ...$**, or** `is_zero` **for** $... \supset f(...) = 0$
- **Even mundane things like direction of equality is important (i.e. want to avoid** `(rewrite ''blah'' :direction RL)`

Mathematical Generality And Elegance Versus Simplicity And Ease Of Use

The NASA PVS library contains three vectors libraries

1. **2-dimensional vectors**
2. **3-dimensional vectors**
3. **N-dimensional vectors**

Why include 2D and 3D versions when an N-dimensional version is available?

ANSWER: Because there are some notational and proof conveniences for doing so.

However, the differences in the libraries are kept to a minimum:

- **same operators**
- **same lemma names (as much as possible)**
- **same theory structure**

Vectors Libraries (cont.)

For example, in the 2D version we represent a vector as

```
Vector: TYPE = [#  x, y: real  #]
```

whereas in the N-dimensional library a vector is

```
Index      : TYPE = below(n)
Vector     : TYPE = [Index -> real]
```

where `n` is a formal parameter (`posnat`) to the theory.

Thus, in the two dimensional case, the x-component of a vector `v` is `` v`x `` whereas in the N-dimensional library it is `v(0)`.

Also certain operations are greatly simplified in the 2D case. The dot product is

```
*(u,v): real  =  u`x * v`x + u`y * v`y;              % dot product
```

in the 2-dimensional case, whereas in the N-dimensional case it is

```
*(u,v): real = sigma(0,n-1,LAMBDA i:u(i)*v(i));     % dot Product
```

where sigma is a summation operator imported from the reals library.

2D Vectors

The vector operators are defined as follows:

```
a      : VAR real
u,v,w  : VAR Vector

-(v)   : Vector = (-v`x, -v`y);
+(u,v): Vector = (u`x + v`x, u`y + v`y);
-(u,v): Vector = (u`x - v`x, u`y - v`y);
*(u,v): real   =  u`x * v`x + u`y * v`y;          % dot product
*(a,v): Vector = (a * v`x, a * v`y);
```

NOTE: A conversion is provided so that one can create 2D vectors as follows

```
(xv,yv)
```

rather than having to write

```
(# x := xv, y := yv #)
```

Distance in 2D space

The theory `positions2D` enhances the vector space with constructs for specifying distances.

```
sq_dist(p1,p2: Pos2D): nnreal = sq(p1`x - p2`x) + sq(p1`y - p2`y)
dist(p1,p2: Pos2D)   : nnreal = sqrt(sq_dist(p1,p2))
```

Many lemmas are available, including

```
dist_refl     : LEMMA dist(p,p) = 0
dist_sym      : LEMMA dist(p1,p2) = dist(p2,p1)
dist_eq_0     : LEMMA dist(p1,p2) = 0 IFF p1 = p2
dist_norm     : LEMMA dist(u,v) = norm(u-v)
sq_dist_le    : LEMMA sq_dist(v1,v2) <= sq_dist(p1,p2) IMPLIES
                           dist(v1,v2) <= dist(p1,p2)
dist_ge_x     : LEMMA dist(p1,p2) >= abs(p1`x - p2`x)
dist_ge_y     : LEMMA dist(p1,p2) >= abs(p1`y - p2`y)
dist_triangle: LEMMA sq(dist(p2,p0)) = sq(dist(p1,p0)) + sq(dist(p1,p2))
                                         - 2*(p1-p0)*(p1-p2)
```

Should I have developed the notion of a general metric space first and imported that rather than just define "dist"?

Completeness Of Theories Verses How Easy To Locate Things

If you add more

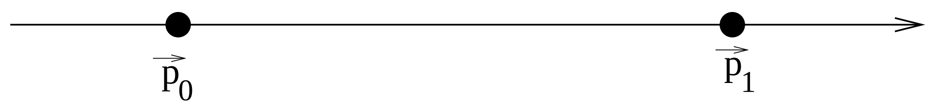
- **You are less likely to hear "Why didn't you include ...."**
- **or "The approach given in the classic text ... is much better"**

BUT

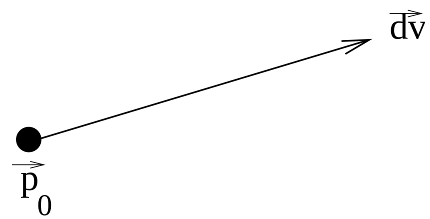
- **the theories get large**
- **there are different versions of same lemma**
- **lots of clutter**
- **more difficult to find what you are looking for**

Example: How many ways can you think of to specify a line in 2D?

- $y = mx + b$
- $f(x, y) = ax + by + c = 0$
- **By specifying two distinct points, $\vec{p_0}$ and $\vec{p_1}$, on it:**



- **By a point and a direction. Let $\vec{p_0}$ be a point on the line L and let $\vec{dv}$ be a nonzero vector specifying the direction of the line.**



- **Should we add dynamics to our line:**

$$\vec{p_0} + t * \vec{\mathtt{vel}}$$

which provides the location of the particle at time t that is traveling with a constant velocity.

My Decision: 2D Lines

In the library, (i.e. theory `lines2D`) lines are defined as a tuple:

```
                                       %     Basic          |   Dynamic
                                       %-------------------|-----------------
  Line: TYPE = [# p: Vect2,            % point on the line| position at time 0
                  v: Nz_vect2 #]       % direction vector | velocity vector
```

This enables one to represent a line using a point and a direction vector

```
     p(L) + v(L)                  or              L`p + L`v
```

or using a point and a velocity vector

```
     p(L) + t * v(L)              or              L`p + t * L`v
```

Multiple definitions – which one?

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

or

$$f'(x) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

- **How about both?**

How Many Lemmas? Which Ones?

Example: Intersecting Lines

The theory `intersections2D` provides some efficient methods for finding intersections using a function named `cross`:

$$\texttt{cross}(\vec{p}, \vec{q}) = p_x * q_y - q_x * p_y$$

The following simple property hold for `cross`:

$$\texttt{cross}(\vec{p}, \vec{q}) = -\texttt{cross}(\vec{q}, \vec{p})$$

There are three cases for two lines $L0$ and $L1$:

intersecting:	$\texttt{cross}(L0_v, L1_v) \neq 0$
parallel:	$\texttt{cross}(L0_v, L1_v) = 0$ **AND** $\texttt{cross}(\Delta, L0_v) \neq 0$
same line:	$\texttt{cross}(L0_v, L1_v) = 0$ **AND** $\texttt{cross}(\Delta, L0_v) = 0$

where $\Delta = L1_p - L0_p$.

Which Lemmas: Intersecting Lines (cont.)

Correspondingly, the library provides the following predicates:

```
intersect?(L0,L1): bool = cross(L0`v,L1`v) /= 0

same_line?(L0,L1): bool = LET DELTA = L1`p - L0`p IN
                          cross(L0`v,L1`v) = 0 AND cross(DELTA,L0`v) = 0
```

Given two lines that intersect the function `intersect_pt` **returns the intersection point:**

```
intersect_pt(L0:Line2D,L1: Line2D | cross(L0`v,L1`v) /= 0): Pos2D =
                            LET DELTA = L1`p - L0`p,
                               ss = cross(DELTA,L1`v)/cross(L0`v,L1`v) IN
                               L0`p + ss*L0`v
```

Which Lemmas: Intersecting Lines (cont.)

Following lemmas are provided:

```
intersection_lem     : LEMMA cross(L0`v,L1`v) /= 0 IMPLIES
                             LET DELTA = L1`p - L0`p,
                                ss = cross(DELTA,L1`v)/cross(L0`v,L1`v),
                                tt = cross(DELTA,L0`v)/cross(L0`v,L1`v)
                             IN
                          L0`p + ss*L0`v =  L1`p + tt*L1`v

pt_intersect         : LEMMA on_line?(p,L0) AND on_line?(p,L1) AND
                             NOT same_line?(L0,L1) IMPLIES
                                 intersect?(L0,L1)

intersect_pt_unique  : LEMMA intersect?(L0,L1) IMPLIES
                             pnot /= intersect_pt(L0,L1) AND
                             on_line?(pnot,L0)
                          IMPLIES
                             NOT on_line?(pnot,L1)
```

```
same_line_lem        : LEMMA p0 /= p1 AND
                           ( on_line?(p0,L0) AND on_line?(p0,L1) AND
                             on_line?(p1,L0) AND on_line?(p1,L1) )
                                IMPLIES same_line?(L0,L1)

not_same_line        : LEMMA on_line?(p,L0) AND
                            NOT on_line?(p,L1)
                         IMPLIES
                             NOT same_line?(L0,L1)

 intersect_pt_lem     : LEMMA NOT same_line?(L0,L1) AND
                            on_line?(pnot,L0)  AND
                            on_line?(pnot,L1)
                         IMPLIES
                             intersect_pt(L0,L1) = pnot
```

- **There is plenty of redundancy here**
- **I added the ones that came up during proofs for SATS project**
- **Is there a better way?**

Which rewrites do you make automatic?

```
AUTO_REWRITE+ add_zero_left          % zero + v = -v
AUTO_REWRITE+ add_zero_right         % v + zero = v
AUTO_REWRITE+ sub_zero_left          % zero - v = -v
AUTO_REWRITE+ sub_zero_right         % v - zero = v
AUTO_REWRITE+ sub_eq_zero            % u - v = zero IFF u = v

AUTO_REWRITE+ sub_eq_args            % v - v = zero
AUTO_REWRITE+ neg_add_left           % -v + v = zero
AUTO_REWRITE+ neg_add_right          % v + -v = zero
AUTO_REWRITE+ dot_zero_left          % zero * v = 0
AUTO_REWRITE+ dot_zero_right         % v * zero  = 0
AUTO_REWRITE+ scal_zero              % a * zero = zero
AUTO_REWRITE+ scal_0                 % 0 * v = zero
AUTO_REWRITE+ sqv_zero               % sqv(zero) = 0
AUTO_REWRITE+ add_neg_sub            % v + -u = v - u
AUTO_REWRITE+ neg_neg                % --v = v
AUTO_REWRITE+ dot_scal_left          % (a*u)*v = a*(u*v)
AUTO_REWRITE+ dot_scal_right         % u*(a*v) = a*(u*v)
AUTO_REWRITE+ dot_scal_assoc         % a*(b*u) = (a*b)*u
AUTO_REWRITE+ dot_scal_canon         % (a*u)*(b*v) = (a*b)*(u*v)
AUTO_REWRITE+ sqv_neg                % sqv(-v) = sqv(v)
AUTO_REWRITE+ sqrt_sqv_sq            % sqrt(sqv(v)) * sqrt(sqv(v)) = sqv(v)
AUTO_REWRITE+ norm_neg               % norm(-u)  = norm(u)
AUTO_REWRITE+ comp_zero_x            % zero`x = 0
AUTO_REWRITE+ comp_zero_y            % zero`y = 0
```

```
    AUTO_REWRITE+ add_cancel             % v + w - v = w
    AUTO_REWRITE+ sub_cancel             % v - w - v = -w
    AUTO_REWRITE+ add_cancel_neg         % -v + w + v = w
    AUTO_REWRITE+ sub_cancel_neg         % -v - w + v = -w
    AUTO_REWRITE+ add_cancel2            % w - v + v = w
    AUTO_REWRITE+ add_cancel_neg2        % w + v - v = w

%   ---- Turn off dangerous and unhelpful rewrites for auto-rewrite-theory --

    AUTO_REWRITE-    add_comm            % u+v = v+u
    AUTO_REWRITE-    dot_comm            % u*v = v*u
    AUTO_REWRITE-    dot_assoc           % a*(v*w) = (a*v)*w
    AUTO_REWRITE-    sqv_lem             % sqv(u) = u*u
    AUTO_REWRITE-    sqv_sym             % sqv(u-v) = sqv(v-u)
    AUTO_REWRITE-    norm_sym            % norm(u-v) = norm(v-u)
    AUTO_REWRITE-    dot_sq_norm         % u*u = sq(norm(u))
```

More Auto-Rewriting

```
(defstep vect-distr ()
    (auto-rewrite "vx_distr_add" "vy_distr_add" "vz_distr_add" "vx_distr_sub"
                  "vy_distr_sub" "vz_distr_sub" "vx_scal" "vy_scal" "vz_scal"
                  "sqv_scal" "norm_scal"
                  "neg_distr_sub" "neg_distr_add" "dot_distr_add_left"
                  "dot_distr_add_right " "dot_distr_sub_left"
                  "dot_distr_sub_right" "dot_scal_distr_add"
                  "dot_scal_distr_sub" "scal_add_distr" "scal_sub_distr"
                  "vx_scal" "vy_scal")
"Turns on auto-rewrites for vectors that apply distributive properties."
"~%Distributive properties for vectors added as auto-rewrites")

(defstep vect-distr-off ()
    (stop-rewrite "vx_distr_add" "vy_distr_add" "vz_distr_add" "vx_distr_sub"
                  "vy_distr_sub" "vz_distr_sub" "vx_scal" "vy_scal" "vz_scal"
                  "sqv_scal" "norm_scal"
                  "neg_distr_sub" "neg_distr_add" "dot_distr_add_left"
                  "dot_distr_add_right " "dot_distr_sub_left"
                  "dot_distr_sub_right" "dot_scal_distr_add"
                  "dot_scal_distr_sub" "scal_add_distr" "scal_sub_distr"
                  "vx_scal" "vy_scal")
"Turns off auto-rewrites for vectors that apply distributive properties."
"~%Distributive propertieauto-rewrites for vectors turned off")
```

Still More Auto-Rewriting

IMPORTING vectors2D_rew

```
vectors2D_rew : THEORY

%
%   Additional more aggressive rewrites

BEGIN

  IMPORTING vectors2D

  u,v,w    : VAR Vect2
  a,b      : VAR real

  AUTO_REWRITE+ vx_distr_add
  AUTO_REWRITE+ vy_distr_add
  AUTO_REWRITE+ vx_distr_sub
  AUTO_REWRITE+ vy_distr_sub
  AUTO_REWRITE+ vx_scal                %% NEW %%
  AUTO_REWRITE+ vy_scal                %% NEW %%

  AUTO_REWRITE+ sqv_scal
  AUTO_REWRITE+ norm_scal

  AUTO_REWRITE+ neg_distr_sub
  AUTO_REWRITE+ neg_distr_add
  AUTO_REWRITE+ dot_distr_add_left
```

Conclusion

- Many, many issues and tradeoffs
- Very little objective guidance on how to make them
- If I were to develop a naming glossary, would people adhere to it?
- What if an illustrious committee reviewed/revised it?
- What other things could be done to make libraries easier to use and work together?