Developing and Proving Algorithms with PVS (Part III)

César Muñoz

cesar.a.munoz@nasa.gov

Fifteenth Summer School on Formal Techniques

May 23–29, 2026

Design Principles for the PVS Practitioner

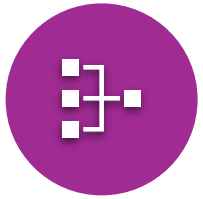
1. Specify for the current system by thinking about the next one (but don’t overthink it)
2. Specify for proofs, first, and animations, second
3. Specify, animate, prove, in that order
4. Prove with robustness and maintenance in mind
5. Specifications and proofs should not be longer than needed

PVS Tips: Recommendations for the PVS Practitioner (In No Particular Order)

Use weak types in the domain and strong types in the range.

Be wary of constants as theory parameters.

Put everything that is important about a function in its type.



Separate definitions, properties, and animation artifacts in different theories.

Grind is your friend and your enemy.

Proof should not be longer than 1000 steps.

Prove TCCs before other properties and before animations.

Use labels and names in proofs.

PVS Tip: Use weak types in the domain and strong types in the range

- Note that $[A_1 \rightarrow B_1]$ is a subtype of $[A_2 \rightarrow B_2]$ **only** when
  - $A_1 = A_2$ and $B_1 < B_2$
  - i.e., no variant and no contra-variant semantics
- **In particular, set[(even?)] is not a sub-type of set[nat]**
- Always define set[T] on the "weakest" T and use predicate sub-typing for restricting, e.g., use {n : nat | even?(n) AND n > 2 } instead of {n : (even?) | n > 2}

PVS Tip: Be wary of constants as theory parameters

- If you think you need a constant as theory parameter, think twice.
- Often inconvenient when importing theory with different constant values.
- May create unnecessary dependencies on definitions and lemmas.
- Even if you need it, prenex-polymorphism may be a better alternative, e.g.,
  MyType[n:nat] : TYPE = [below(n) →T]

PVS Tip: Put everything that is important about a function in its type

- You should be able to prove high-level properties about a function without unfolding.
- Use JUDGEMENTs to refine the domain and range of a function.
- Use RECURSIVE JUDGEMENTs to refine the domain and range of a recursive function (and get induction-free inductive proofs).

PVS Tip: Separate definitions, properties, and animation artifacts in different theories

- If you need to import a theory to prove a property of a function, that property should be in a different theory as the definition.
- This way, other theories could import only the definitions without the extra burden of importing other external theories.
- Put all definitions that are needed for PVSio animations in a separate theory.

PVS Tip: Grind is your friend and your enemy

- If (grind) doesn’t terminate in less than 10s, stop it.
- If (grind) doesn’t discharge the current goal, undo it.
- (grind) has many options, use them, e.g., (grind :exclude …).
- (grind) is not an efficient evaluator. If formula is ground, use (eval-formula …)
- Try (prop), (assert), (ground) before (grind), in that order.
- Use (grind-reals), if formula involves real-valued expressions.