Validating Critical Systems With PVS

Mariano Moscato
mariano.moscato@nianet.org

National Institute of Aerospace

2018

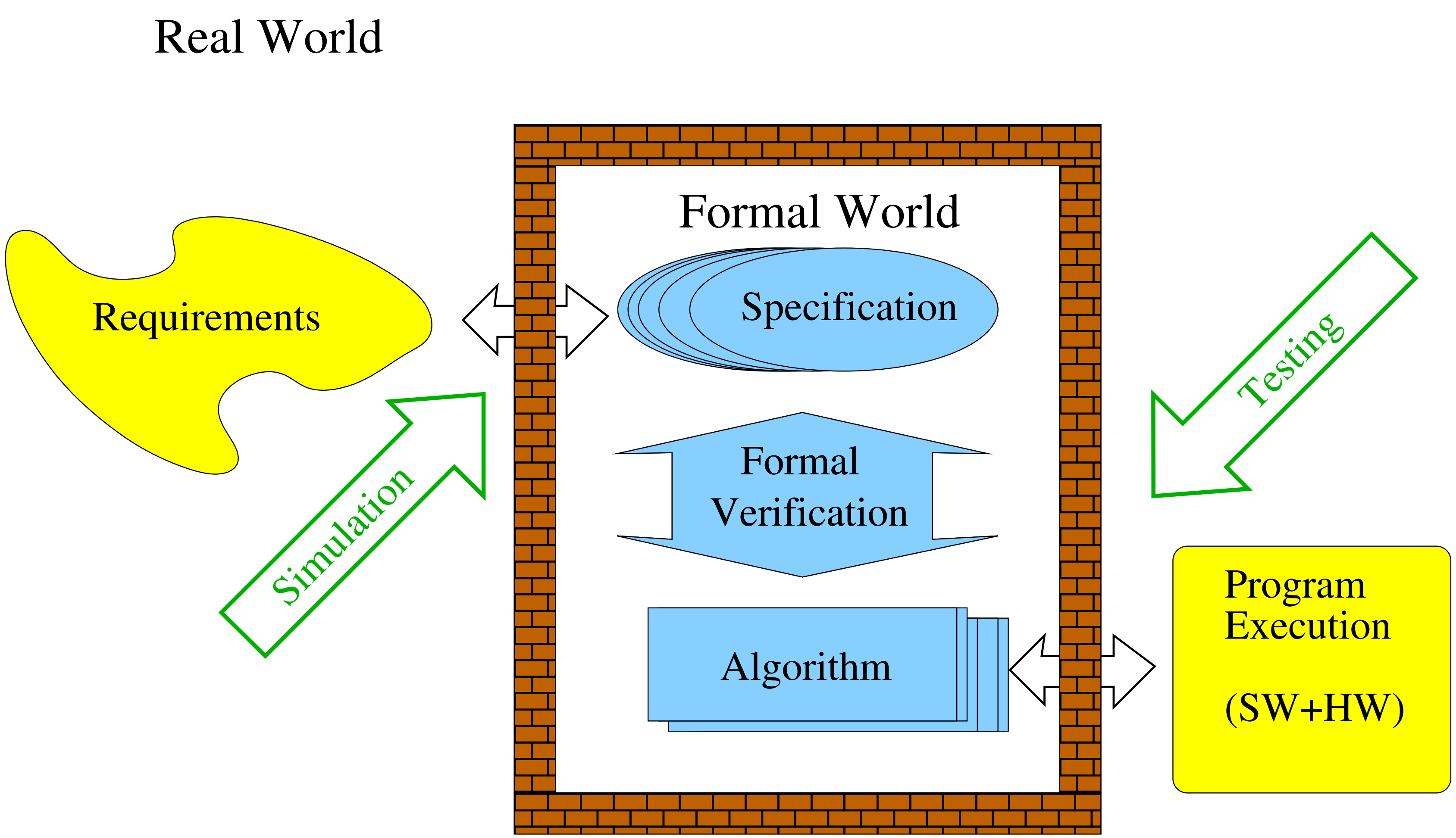
Nice To Meet You!

- BS and PhD in Computer Science, University of Buenos Aires
  - Thesis: "Improvements to Interactive Theorem Proving of Alloy Properties using SAT-Solving." (Adv.: Marcelo Frias)
  - Introduced to PVS in late 2006
- Join National Institute of Aerospace in 2014
- Part of the NASA Langley Formal Methods Group since 2014
  - Working mainly with PVS since then
- No lectures since then… sorry!

PVS

- Prototype Verification System (http://pvs.csl.sri.com).
- Developed by SRI International (http://www.sri.com).
- 1992: Version 1. Currently version 6.
  - On the horizon: version 7.
- Strongly typed specification language based on classical higher-order logic.
- Theorem prover with built-in decision procedures.

In Which Ways Do We Use PVS?



- Specification
- Correcteness proofs
- Validation of systems

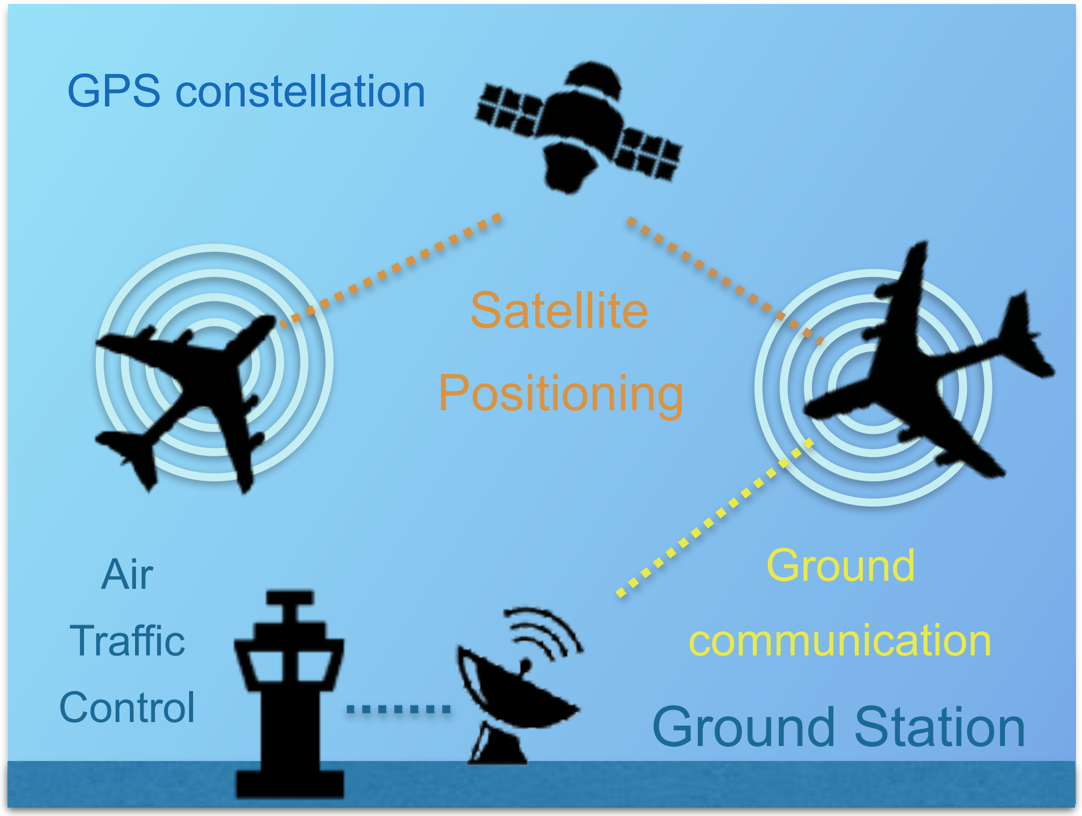
- Safety- and mission-critical systems
- Avionics

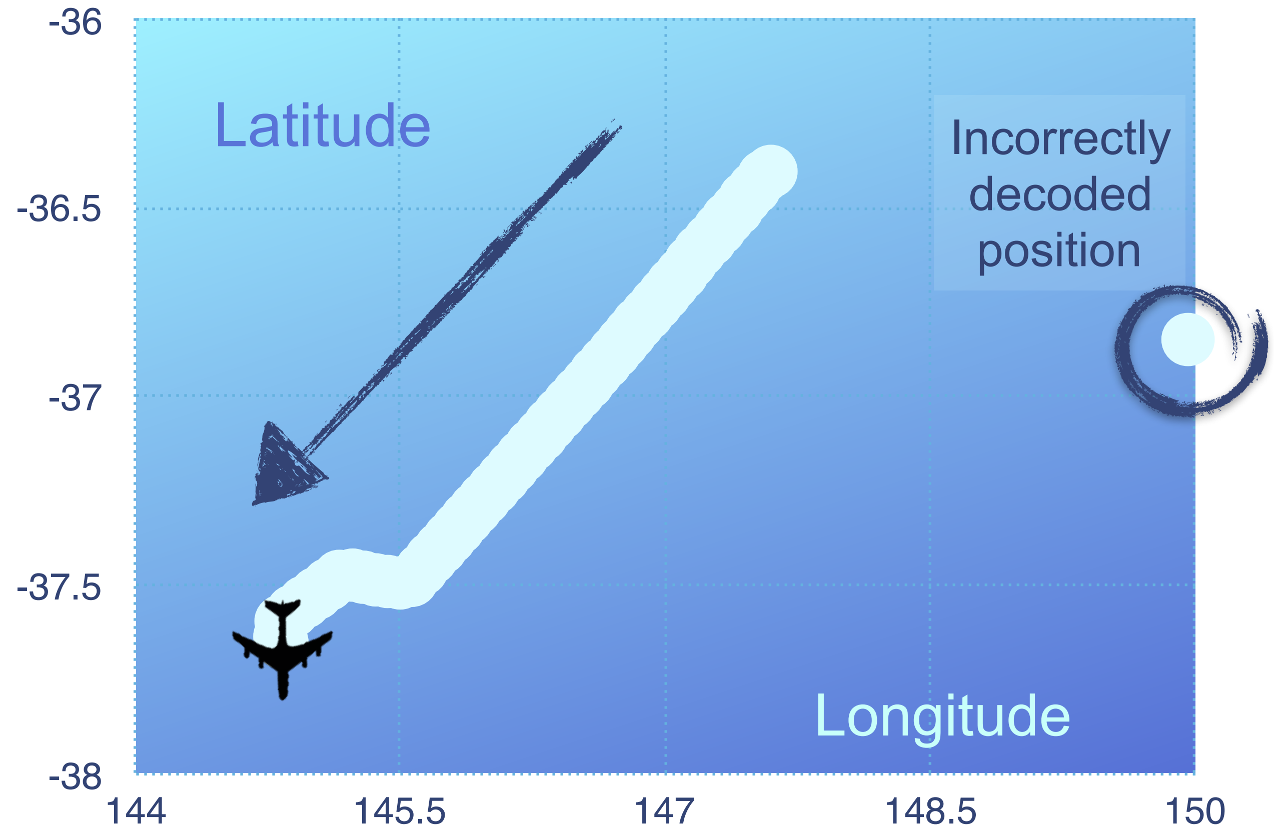
Some Examples

- Rigorous approximation of non-linear functions
  - Affine Arithmetic
- Bug finding in avionics systems
  - Compact Position Reporting Algorithm
- PRECiSA: Program Round-off Error Certifier via Static Analysis
  - Floating-point program analyzer
- Practical tools to improve provers
  - and the life quality of the humans using them

CPR

- Automatic Dependent Surveillance - Broadcast (ADS-B) protocol
  - Mandatory on Jan 1, 2020 (FAA) in USA and Europe
- Compact Position Reporting Algorithm





- Result: A tightened set of requirements, a simplified algorithm
  - will be included in the next version of the standard by the competent organizations.

What Do I Expect From This Course

- My Goal:
  - From the next week on, you will be able to state and prove properties using PVS
- Strategy of the class:
  - based on your participation/interests/background,
  - short lectures,
  - hands-on sessions
- We are (always) looking for people trained (and interested) in working with PVS
  - Formal Methods in general
- Exam: every day exercises + final exercise

A Gentle Introduction To EMACS

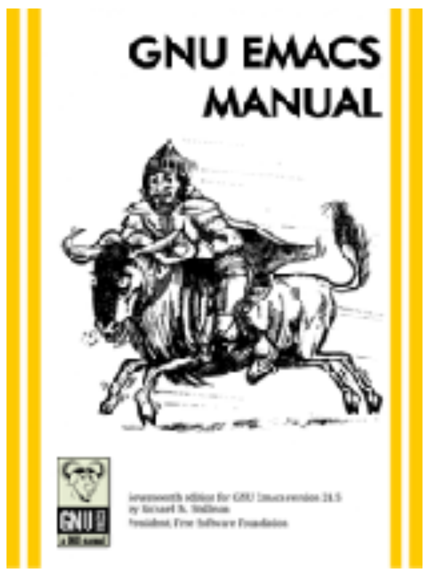
Mariano M. Moscato
mariano.moscato@nianet.org

National Institute of Aerospace

Feb. 2018

A Gnu Ride for PVS

- EMACS: a highly customizable editor



  - *"extensible, customizable, self-documenting real-time display editor"* according to the manual[1]

- In indeed, it is the (main) UI for PVS
- Sorry if you love text-based interfaces, I'm about to be mean…

Hint: do not stand in the way of this gnu…

1. https://www.gnu.org/software/emacs/emacs.html

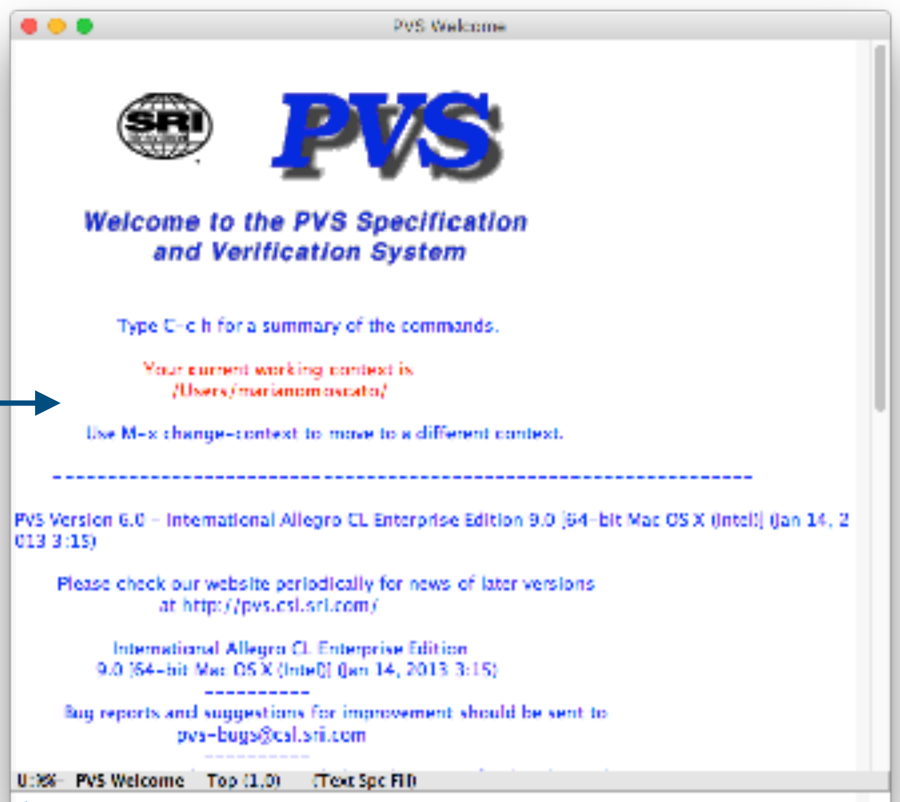
A Gnu Ride for PVS

GNU Emacs @ Ubuntu

Aquamacs @ OSX

Window

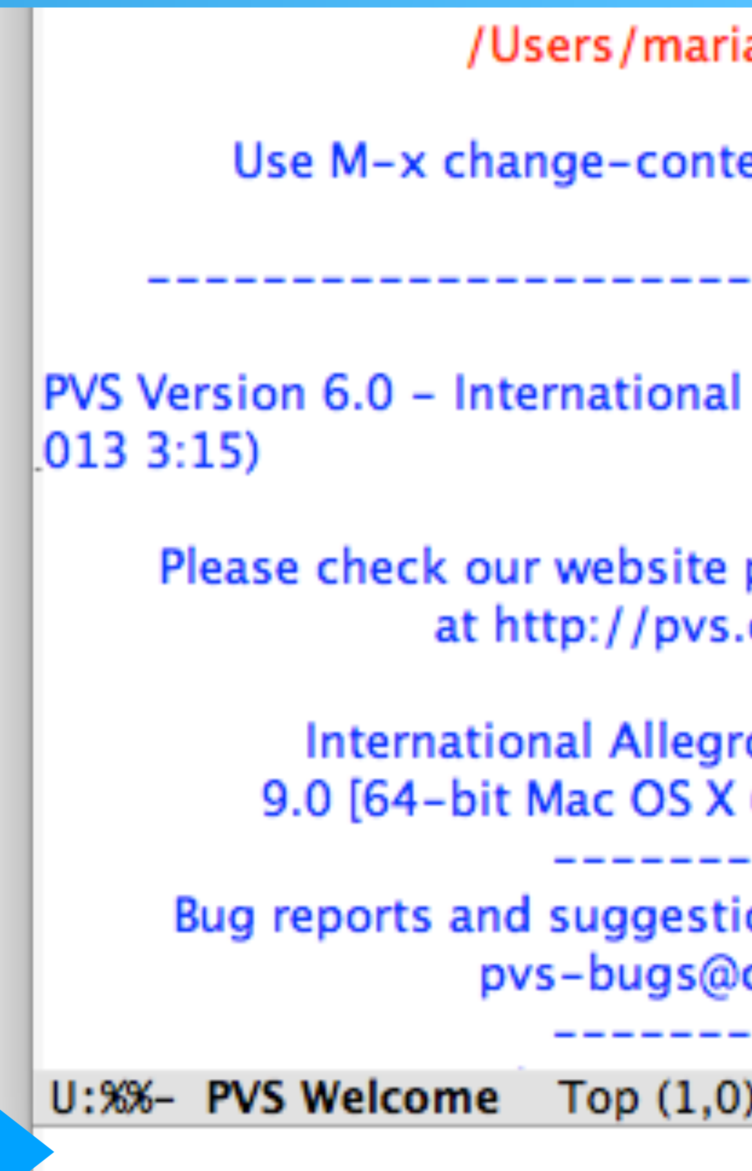
Buffer



Minibuffer

Commands

- Every command in Emacs has a name
  - convention: one of more words separated by a hyphen
  - for example: `find-file` is used to open a file
- To run a command:
  - `<ALT> x <command name> <RET>`
  - `<ALT> x` allows access to the Minibuffer, where commands are issued



- There are, of course, a LOT of shortcuts…

Your New Two Friends

C-<chr>

It means Ctrl + <key>
Also knows as ^<chr>

M-<chr>

It means Alt + <key>
("Meta" key)

- For example:
  - C-x C-f is the shortcut to open/create a file
- Warning: Copy-Paste is not as usual!
  - M-w copy
  - C-w cut
  - C-y paste (yank)

- M-f moves the cursor forward by a word

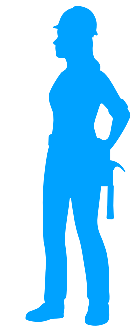
- C-v page down
- C-x C-s exit Emacs
- C-c nothing (by its own)

Starting PVS

- Shell command "pvs"
  - $ pvs
- PVS works with so-called "contexts"
  - automatically assigned to the folder where PVS was started
  - M-x context-path shows the current context path
  - M-x change-context can be used to change the context

Exercise 0

- Make a directory named "day1" and cd to it
- Open PVS
- Create a new file named "hello.txt"
  - C-x C-f + hello.txt
- Copy and Paste contents of "template.txt" to your file
- Fill the blanks
- Save file
  - C-x C-s
- Send it to mariano.moscato@nianet.org



Prototype Verification System

Mariano M. Moscato
mariano.moscato@nianet.org

National Institute of Aerospace

Outline

- PVS Language
- Basic Declarations
- Types
- Formula Declarations
- Proving Properties

Note

These slides are based on the PVS manuals

(by Shankar, Owre, Rushby, Stringer-Calvert)

- Publicly available at http://pvs.csl.sri.com/documentation.shtml
- In particular:
  - Language Reference
  - Prover Guide
  - Prelude

PVS Language

PVS Language

- PVS language is based on strongly-typed higher-order logic.
  - Higher-order logic:
    - in first-order logic it is possible to talk about all human kind
    - in higher-order logic it is possible to talk about every possible group of humans
  - Strongly-Typed:
    - every expression has (at least one) type
- Terms can be constructed using
  - function application,
  - lambda abstraction, and
  - record and tuple construction.

About PVS's Type System

- PVS supports *predicate subtyping*
- Predicate Subtype:
  - the subset of individuals in a type satisfying a given predicate
  - Example: {x:real | x /= 0}
- Typechecking is undecidable
  - since the predicate used in defining a predicate subtype is arbitrary,
  - it produces proof obligations called Type Correctness Conditions (TCCs)
  - the user is expected to discharge these proof obligations
  - Typechecking cannot be considered complete until all TCCs have been proved

PVS Language

- A PVS specification consists of a collection of *theories*
- Each theory consists of
  - a signature for the type names and constants introduced in the theory, and
  - the axioms, definitions, and theorems associated with the signature.

PVS Prelude

A large body of theories providing:

- infrastructure for the PVS typechecker and prover, and
- definitions supporting the specification and verification of systems

Can be accessed through the commands

- M-x view-prelude-file / M-x view-prelude-theory
- **Try**: M-x view-prelude-theory booleans

Booleans Theory

```
booleans: THEORY
 BEGIN

  boolean: NONEMPTY_TYPE
  bool: NONEMPTY_TYPE = boolean
  FALSE, TRUE: bool
  NOT, ¬: [bool -> bool]
  AND, &, ∧, OR, ∨, IMPLIES, =>, ⇒, WHEN, IFF, <=>, ⇔: [bool, bool -> bool]

 END booleans
```

Declarations:

- Types: boolean, bool
- Constants: FALSE, TRUE
- Predicates/functions: (unary) NOT, ¬, (binary) AND, &, ∧, OR, ∨, IMPLIES […]

Types

- The PVS type system is based on structural equivalence
  - instead of name equivalence,
  - types are closely related to sets,
  - two types are equal iff they have the same elements.
- Basic distinction
  - Uninterpreted types `boolean: NONEMPTY_TYPE`
  - Interpreted types `bool: NONEMPTY_TYPE = boolean`

Basic Declarations

Constant Declarations

- Constant declarations introduce new constants
- specifying their type and optionally providing a value
- *constant* refers to functions, relations, and regular (0-ary) constants:

```
n: int
c: int = 3
f: [int -> int] = (lambda (x: int): x + 1)
g(x: int): int = x + 1
```

- Constants can have (either) uninterpreted or interpreted declarations
- No assumptions are posed on uninterpreted constants
  - besides PVS requires the type of the constant to be nonempty

Recursive Definitions

- PVS allows a restricted form of recursive definition
  - mutual recursion is not allowed
- The recursive function must be total
  - The user must provide a measure and an optional well-founded order relation
- Measure
  - a function that can be used to prove that the input decreases on each recursion
  - A TCC is generated by the typechecker stating such proof obligation
  - its signature matches that of the function, but its range is the domain of the order relation

Recursive Definitions

```
factorial(x: nat): RECURSIVE nat =
  IF x = 0 THEN 1 ELSE x * factorial(x - 1) ENDIF
  MEASURE (LAMBDA (x: nat): x)
```

- The measure is the expression following the MEASURE keyword
  - In this case, the identity function
- This definition generates the following *termination TCC*:

  factorial_TCC2: OBLIGATION FORALL (x: nat): NOT x = 0 IMPLIES x - 1 < x
- To show the TCCs: M-x show-tccs-theory (or M-x tccs or C-c C-q s)
- A termination TCC is generated for each recursive occurrence of the defined entity within the body of the definition.