

Real Applications

Formal Methods for the Next Generation of Air Traffic Management Systems

César A. Muñoz

NASA Langley Research Center
Cesar.A.Munoz@nasa.gov



Outline

The Airspace System

2D Conflict Detection and Resolution (CDR2D)

Real Numbers in PVS

Formal Development of CDR2D

Real Number Proving Tools

- Basic Manipulations via Manip

- Simplifications via Field

- Interval Arithmetic via Interval

- Solving Polynomial Inequalities via Bernstein

Concluding Remarks

The Airspace System

- ▶ 24h of air traffic in the USA
- ▶ 24h of air traffic in the World

Next Generation Air Transportation System (NextGen)

- ▶ FAA and industry forecast a **1.5 – 2.5 times increase** of air traffic operations over the next two decades.
- ▶ *By 2016, develop and demonstrate future concepts, capabilities, and technologies that will enable **major increases** in air traffic management . . . while **maintaining safety**, to meet capacity and mobility requirement of NextGen.*

Enabling Technology

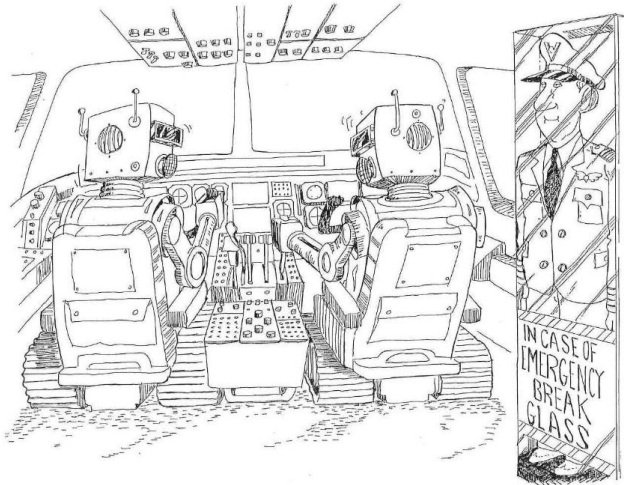
- ▶ Hardware Systems:
 - ▶ Global Positioning System (GPS).
 - ▶ Automatic Dependent Surveillance-Broadcast (ADS-B).
 - ▶ Increase in computational power.

- ▶ Operational concepts based on distributed and autonomous systems.

Distributed Air/Ground Air Traffic Management

- ▶ Today: Air traffic controllers have the primary responsibility for en-route aircraft separation
- ▶ Future: The responsibility for separation will be **air/ground distributed**.

Automated Cockpit



State-Based Separation Assurance Systems (SAS)

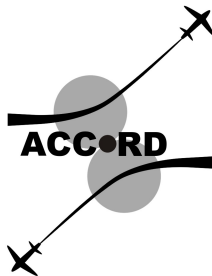
Airborne systems that maintain aircraft **separated** assuming only **state information** (current position and velocity vector), e.g., **Conflict Detection and Resolution (CD&R)**.

SAS Safety Properties

- ▶ Conflict Detection:
 - ▶ *correctness*: No false alerts.
 - ▶ *completeness*: No missed alerts.
- ▶ Conflict Resolution:
 - ▶ *completeness*: At least one solution.
 - ▶ *independence*: Conflict solved when one aircraft maneuvers.
 - ▶ *(Implicit) coordination*: Conflict solved when both aircraft maneuver.

Airborne Coordinated Conflict Resolution and Detection Framework

ACCoRD is a mathematical framework, fully developed in PVS, for the design and verification of state-based separation assurance algorithms.

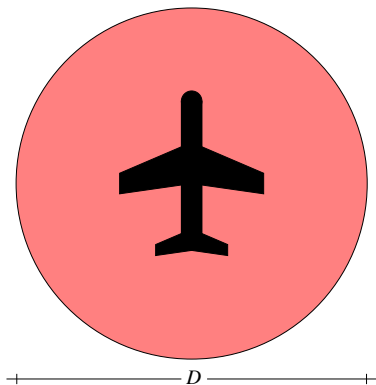


2D Conflict Detection and Resolution (CDR2D)

- ▶ 2D Rectangular coordinate system.
- ▶ Linear trajectories.
- ▶ Instantaneous velocity changes.
- ▶ Perfect state information (position and velocity vector).
- ▶ Perfect communication (broadcast from traffic aircraft).
- ▶ Pairwise approach: *ownership* and *intruder* aircraft.
- ▶ Distributed approach: No central control.
- ▶ Independent decision making: aircraft do not communicate/negotiate their resolution maneuvers.

Basic Definitions: Protected Zone

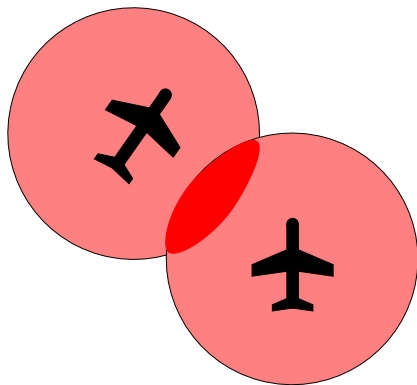
Circular area of radius D around each aircraft. The value D defines a minimum horizontal distance between aircraft.



(Typically, $D = 5nm$)

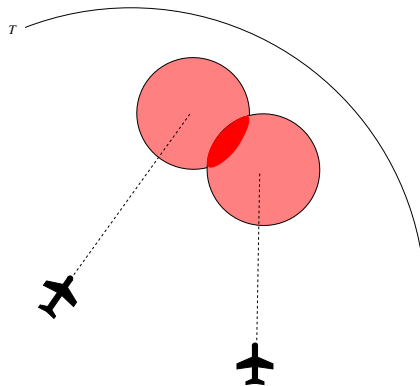
Basic Definitions: Loss of Separation

Violation of separation minima, i.e., overlapping of protected zones.



Basic Definitions: Conflict

Predicted loss of separation within a lookahead time T .



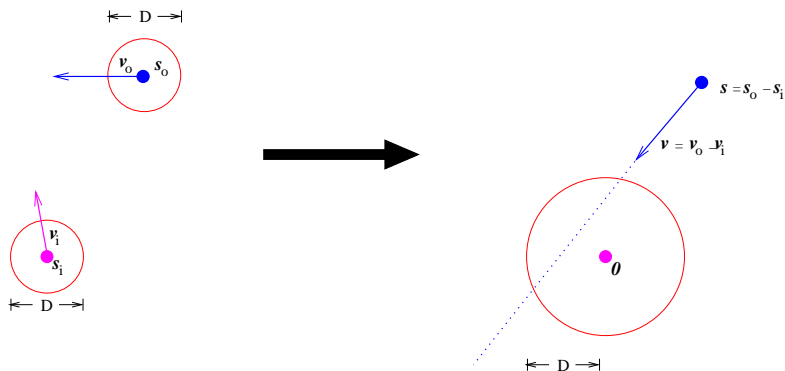
(Typically, $T = 5min$)

Geometry

- ▶ Ownship current state is given by \mathbf{s}_o (position) and \mathbf{v}_o (velocity) in \mathbb{R}^2 .
- ▶ Intruder current state is given by \mathbf{s}_i (position) and \mathbf{v}_i (velocity) in \mathbb{R}^2 .
- ▶ Projected states at time t : $\mathbf{s}_o + t\mathbf{v}_o$ and $\mathbf{s}_i + t\mathbf{v}_i$.
- ▶ Relative view: $\mathbf{s} \equiv \mathbf{s}_o - \mathbf{s}_i$ and $\mathbf{v} \equiv \mathbf{v}_o - \mathbf{v}_i$.

Transformation to Relative Coordinate System

In the relative coordinate system, the intruder fixed at the origin and the ownship moves relative to the intruder.



Conflict

Given a relative position \mathbf{s} and relative velocity \mathbf{v} , there is a (predicted) **conflict** if there exists a time $0 \leq t \leq T$ such that the aircraft are in loss of separation at time t :

$$\text{conflict?}(\mathbf{s}, \mathbf{v}) \equiv \exists t \in [0, T] : \|\mathbf{s} + t\mathbf{v}\| < D.$$

Conflict Detection Algorithms

A **conflict detection algorithm** is a function

$cd(\mathbf{s}_o, \mathbf{v}_o, \mathbf{s}_i, \mathbf{v}_i) : \text{bool}$,

where

- ▶ $\mathbf{s}_o, \mathbf{v}_o$ is current state of the ownship.
- ▶ $\mathbf{s}_i, \mathbf{v}_i$ is the current state of the intruder.

Conflict Detection Correctness and Completeness

- ▶ An algorithm cd is **correct** if it does not have missed alerts, i.e.,

$$conflict?(s_o - s_i, v_o - v_i) \implies cd(s_o, v_o, s_i, v_i).$$

- ▶ An algorithm cd is **complete** if it does not have false alerts, i.e.,

$$cd(s_o, v_o, s_i, v_i) \implies conflict?(s_o - s_i, v_o - v_i).$$

Conflict Resolution Algorithms

A **conflict resolution algorithm** is a function

$$\text{cr}(\mathbf{s}_o, \mathbf{v}_o, \mathbf{s}_i, \mathbf{v}_i) : \wp(\mathbb{R}^2),$$

where $\mathbf{v}'_o \in \text{cr}(\mathbf{s}_o, \mathbf{v}_o, \mathbf{s}_i, \mathbf{v}_i)$ is a *resolution maneuver* for the ownship.

Independent Conflict Resolution

An algorithm cr is **independent** if it provides conflict-free resolution maneuvers assuming that only the ownship maneuvers, i.e.,

For all $\mathbf{v}'_o \in cr(\mathbf{s}_o, \mathbf{v}_o, \mathbf{s}_i, \mathbf{v}_i)$,

$$conflict?(\mathbf{s}_o - \mathbf{s}_i, \mathbf{v}_o - \mathbf{v}_i) \implies \neg conflict?(\mathbf{s}_o - \mathbf{s}_i, \mathbf{v}'_o - \mathbf{v}_i).$$

Coordinated Conflict Resolution

Algorithms cr_o and cr_i are (implicitly) **coordinated** if they provide conflict-free resolution maneuvers assuming that both aircraft simultaneously maneuver, i.e.,

For all $\mathbf{v}'_o \in cr_o(\mathbf{s}_o, \mathbf{v}_o, \mathbf{s}_i, \mathbf{v}_i)$, $\mathbf{v}'_i \in cr_i(\mathbf{s}_i, \mathbf{v}_i, \mathbf{s}_o, \mathbf{v}_o)$,

$$conflict?(\mathbf{s}_o - \mathbf{s}_i, \mathbf{v}_o - \mathbf{v}_i) \implies \neg conflict?(\mathbf{s}_o - \mathbf{s}_i, \mathbf{v}'_o - \mathbf{v}'_i).$$

Real Numbers in PVS

- ▶ Reals are defined as an uninterpreted subtype of `number` in the prelude library:

```
real: TYPE+ FROM number
```

- ▶ All numeric constants are `real`:
 - ▶ naturals: $0, 1, \dots$
 - ▶ integers: $\dots, -1, 0, 1, \dots$
 - ▶ rationals: $\dots, -1/10, \dots, 3/2, \dots$

- ▶ Decimal notation is supported: The decimal number **3.141516** is syntactic sugar for the rational number $31416/10000$.

PVS's real numbers are Real

(Rather than floating point numbers)

- ▶ All the **standard properties**: infinite, non-enumerable,
 $\mathbb{N} \subseteq \mathbb{Z} \subseteq \mathbb{Q} \subseteq \mathbb{R}, \dots$
- ▶ **Exact** arithmetic: $1/3 + 1/3 + 1/3 = 1$.
- ▶ The type real is **unbounded**:

```
googol      : real = 10^100
```

```
googolplex : real = 10^googol
```

```
googol_prop : LEMMA
```

```
  googolplex > googol * googol
```


Ground real Arithmetic is Built-in

- ▶ Numerical expressions can be **automatically** reduced by the theorem prover (no need to prove $1+1=2$), ...
- ▶ ... except for *machine physical limitations*, e.g., you probably cannot prove `googol_prop` with `(grind)`.
- ▶ But, you can still prove `googol_prop` using analytical methods.

Subtypes of real

```
nzreal  : TYPE+ = {r:real | r /= 0} % Nonzero reals
nnreal  : TYPE+ = {r:real | r >= 0} % Nonnegative reals
npreal  : TYPE+ = {r:real | r <= 0} % Nonpositive reals
negreal : TYPE+ = {r:real | r < 0} % Negative reals
posreal : TYPE+ = {r:real | r > 0} % Positive reals

rat      : TYPE+ FROM real
int      : TYPE+ FROM rat
nat      : TYPE+ FROM int
```

The uninterpreted type `number` is the only `real`'s supertype predefined in PVS: no complex numbers, no hyper-reals, no \mathbb{R}^∞ , ...

Predefined Operations

```
+ , - , * : [real , real -> real]  
/ : [real , nzreal -> real]  
- : [real -> real]
```

```
sgn(x:real) : int = IF x >= 0 THEN 1 ELSE -1 ENDIF  
abs(x:real) : {nny: nreal | nny >= x} = ...  
max(x,y:real): {z: real | z >= x AND z >= y} = ...  
min(x,y:real): {z: real | z <= x AND z <= y} = ...  
^(x: real,i:{i:int | x /= 0 OR i >= 0}): real = ...
```

... and what about $\sqrt{\quad}$, \int , \log , \exp , \sin , \cos , \tan , π , \lim , ... ?

NASA PVS Libraries

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

- ▶ `reals`: Square, square root, quadratic formula, polynomials.
- ▶ `analysis`: Real analysis, limits, continuity, derivatives, integrals.
- ▶ `vectors` and `vect_analysis`: Vector calculus and analysis.
- ▶ `series`: Power series, Taylor's theorem.
- ▶ `lnexp` and `lnexp_fnd`: Logarithm, exponential, and hyperbolic functions.
- ▶ `trig` and `trig_fnd`: Trigonometry.
- ▶ `complex`: Complex numbers.
- ▶ `float`: Floating point numbers.
- ▶ ...

To Be Or Not To Be (Fundational) ?

- ▶ Axiomatic theories `trig` and `lnexp` typecheck faster.
- ▶ Fundational theories `trig_fnd` and `lnexp_fnd` have **no** axioms.
- ▶ Be careful what you wish for:

```

|-----
{1}  sin(pi / 2) > 1 / 2

```

Rule? (**grind**)

```
Integral rewrites Integral[real](0, 1, atan_deriv_fn)
```

```
  to integral(0, 1, atan_deriv_fn)
```

```
atan_value rewrites atan_value(1)
```

```
  to integral(0, 1, atan_deriv_fn)
```

```
atan rewrites atan(1)
```

```
...
```

Formal Development of CDR2D

```
CDR2D[D:posreal] : THEORY
BEGIN IMPORTING vectors@vectors_2D
  % 2D Positions
  Position : TYPE = Vect2
  p,s,so,si : VAR Position

  % 2D Relative Velocities
  Velocity : TYPE = Vect2
  v : VAR Velocity

  % 2D Absolute Velocities (cannot be zero)
  AbsVelocity : TYPE = Nz_vect2
  vo,vop,vi,vip : VAR AbsVelocity
  ...
END CDR2D
```

Relative Protected Zone

ProtectedZone : `set[Position]` = {p | `sq(p) < sq(D)`}

Sets in PVS are characteristic functions, e.g., the definition above is equivalent to

ProtectedZone(p) : `bool` = `sq(p) < sq(D)`

Conflict, Conflict Detection, and Conflict Resolution

```
conflict?(s,v): bool =  
  EXISTS(t:nnreal|t <= T): ProtectedZone(s+t*v)  
  
% Type of CD algorithms  
CD : TYPE =  
  PRED[[Position,AbsVelocity,Position,AbsVelocity]]  
  
% Type of CR algorithms  
CR : TYPE = [[Position,AbsVelocity,Position,AbsVelocity]  
  -> set[AbsVelocity]]
```

Note: PRED[T] is the same as [T->bool].

CD Correctness and Completeness

```
cd : VAR CD
```

```
correct?(cd) : bool =  
  FORALL(so,vo,si,vi):  
    conflict?(so-si,vo-vi) IMPLIES cd(so,vo,si,vi)
```

```
complete?(cd) : bool =  
  FORALL(so,vo,si,vi):  
    cd(so,vo,si,vi) IMPLIES conflict?(so-si,vo-vi)
```

CR Independence and Coordination

```
cr : VAR CR
```

```
independent?(cr) : bool =  
  FORALL(so,vo,si,vi,vop) :  
    conflict?(so-si,vo-vi) AND  
    member(vop,cr(so,vo,si,vi)) IMPLIES  
    NOT conflict?(so-si,vop-vi)
```

```
coordinated?(cr) : bool =  
  FORALL(so,vo,si,vi,vop,vip) :  
    conflict?(so-si,vo-vi) AND  
    member(vop,cr(so,vo,si,vi)) IMPLIES  
    member(vip,cr(si,vi,si,vi)) IMPLIES  
    NOT conflict?(so-si,vop-vip)
```

A CD Algorithm

```
cd2d_ever?(s,v): MACRO bool =  
  IF s*v < 0 THEN  
    sq(s*v)-sqv(v)*(sqv(s)-sq(D)) > 0  
  ELSE sqv(s) < sq(D)  
  ENDIF
```

```
cd2d_ever(so,vo,si,vi): bool =  
  cd2d_ever?(so-si,vo-vi)
```

```
cd2d_ever_incomplete : THEOREM  
  NOT complete?(cd2d_ever)
```

```
cd2d_ever_correct : THEOREM  
  correct?(cd2d_ever)
```

Exercise 1

1. Define a type of aircraft with an identifier, a 2D position, and a 2D velocity:

```
AircraftXY : TYPE = ...
```

2. Define a constant ac of type AircraftXY:

```
ac : AircraftXY = ...
```

3. Define a predicate cdnas that holds if there is a traffic aircraft that is in conflict ever, e.g., cd2d_ever, with the ownship.

```
cdnas(ownid:Identifier,nas:(exists?(ownid))) : bool =  
...
```

Flat Earth?

```
IMPORTING vectors@ECEF,  
          vectors@vect3_basis,  
          vectors@trackAngles_2D  
  
% Radius of the earth  
R : posreal = 6353000 % [m]  
  
geo2sxy(lat_ref,lon_ref:real)(lat,lon:real) : Position =  
  LET midlat = (lat_ref+lat)/2,  
      midlon = (lon_ref+lon)/2,  
      refxyz = spherical2xyz(R,midlat,midlon),  
      pxyz   = spherical2xyz(R,lat,lon) IN  
  sphere_to_2D_plane(refxyz)(pxyz)  
  
gstrk2vxy(gs:posreal,trk:real): AbsVelocity =  
  v_from(trk,gs)
```

Proving `cd2d_ever` Incompleteness

```
cd2d_ever_incomplete : THEOREM
  NOT complete?(cd2d_ever)
```

Proving cd2d_ever Incompleteness

```
cd2d_ever_incomplete :
```

```
{-1} complete?(cd2d_ever)
```

```
|-----
```

```
Rule? (expand "complete?")
```

Expanding the definition of complete?, this simplifies to:

```
cd2d_ever_incomplete :
```

```
{-1} FORALL (so, vo, si, vi):
```

```
    cd2d_ever(so,vo,si,vi) IMPLIES conflict?(so-si,vo-vi)
```

```
|-----
```

```
Rule? (inst -1 "(-D-T,0)" "(2,0)" "(D,0)" "(1,0)")
```

```
cd2d_ever_incomplete.1 :
```

```
{-1} cd2d_ever((# x:=-D - T, y:=0 #), (# x:=2, y:=0 #),
              (# x:=D, y:=0 #), (# x:=1, y:=0 #))
      IMPLIES
      conflict?((# x:=-D - T, y:=0 #) - (# x:=D, y:=0 #),
               (# x:=2, y:=0 #) - (# x:=1, y:=0 #))
      |-----
```

```
Rule? (grind :exclude "sq")
```

```
{-1} t!1 >= 0
{-2} t!1 <= T
{-3} sq(-D - D - T + t!1) < sq(D)
      |-----
```


Rule? (rewrite "sq_neg" :dir rl)
 Rewriting using sq_neg, matching in *, this simplifies to:
 cd2d_ever_incomplete.1.1 :

```
[-1]  t!1 >= 0
[-2]  t!1 <= T
{-3}  sq(-(-D - D - T + t!1)) < sq(D)
      |-----
```

Rule? (rewrite "sq_lt")
 Rewriting using sq_lt, matching in *,
 This completes the proof of cd2d_ever_incomplete.1.1.

All the other sub-goals are easily discharged by grind,
 decompose-equality, and assert.

Proving cd2d_ever Correctness

```
cd2d_ever_correct : THEOREM  
  correct?(cd2d_ever)
```

Proving cd2d_ever Correctness

After a few strategies, we have two sub-goals:

cd2d_ever_correct.1:

{-1} $(so-si)*(vo-vi) < 0$

[-2] $sq((so-si+t*(vo-vi))'x)+sq((so-si+t*(vo-vi))'y) < sq(D)$

|-----

{1} $sq((so-si)*(vo-vi))+sq(D)*sqv(vo-vi) -$
 $sqv(so-si)*sqv(vo-vi) > 0$

cd2d_ever_correct.2:

[-1] $sq((so-si+t*(vo-vi))'x)+sq((so-si+t*(vo-vi))'y) < sq(D)$

|-----

{1} $(so-si)*(vo-vi) < 0$

{2} $sqv(so-si) < sq(D)$

How to prove these kinds of formulas? Very carefully :-).

Real Number Proving Tools

- ▶ Basic algebraic manipulations via Manip.
- ▶ Simplifications via Field.
- ▶ Interval arithmetic via Interval.
- ▶ Solving polynomial inequalities via Bernstein.

Basic Manipulations via Manip

- ▶ Manip is a package for algebraic manipulations of real-valued expressions.
- ▶ `http://shemesh.larc.nasa.gov/people/bld/manip.html`.
- ▶ The package consists of:
 - ▶ Strategies.
 - ▶ Extended notations for formulas and expressions.
 - ▶ Emacs extensions.
 - ▶ Support functions for strategy developers.
- ▶ **Manip is pre-installed in PVS 5.0.**

Examples

Manip

```
{-1}  x + y <= z
      |-----
```

...

Rule? (mult-by -1 "100")

Multiplying both sides of selected formulas by given term,
this simplifies to:

mp :

```
{-1}  (x + y) * 100 <= z * 100
      |-----
```

...

Examples

Manip

```
{-1} (x + y) * 100 <= z * 100
```

```
|-----
```

```
...
```

```
Rule? (flip-ineq -1)
```

Negating and moving the inequalities in formulas -1,
this simplifies to:

```
mp :
```

```
|-----
```

```
{1} (x + y) * 100 > z * 100
```

```
...
```

A Few Manip Strategies

Strategy	Description
(swap-rel fnums)	Swap sides and reverse relations
(swap! expr-loc)	$x \circ y \Rightarrow y \circ x$
(group! expr-loc LR)	$(x \circ y) \circ z \Rightarrow x \circ (y \circ z)$
(flip-ineq fnums)	Negate and move inequalities
(split-ineq fnum)	Split \leq (\geq) into $<$ ($>$) and $=$

More Strategies

Strategy	Description
<code>(mult-by fnums term)</code>	Multiply formula by term
<code>(div-by fnums term)</code>	Divide formula by term
<code>(move-terms fnum L R tnums)</code>	Move additive terms left and right
<code>(isolate fnum L R tnum)</code>	Isolate additive terms
<code>(cross-mult fnums)</code>	Perform cross-multiplications
<code>(factor fnums)</code>	Factorize formulas
<code>(factor! expr-loc)</code>	Factorize terms
<code>(mult-eq fnum fnum)</code>	Multiply equalities
<code>(mult-ineq fnum fnum)</code>	Multiply inequalities

Extended Formula Notation

▶ Standard

- ▶ *: All formulas.
- ▶ -: All formulas in the antecedent.
- ▶ +: All formulas in the consequent.

▶ Extended (Manip strategies only)

- ▶ $(\hat{\ } n_1 \dots n_k)$: All formulas but n_1, \dots, n_k
- ▶ $(-\hat{\ } n_1 \dots n_k)$: All antecedent formulas but n_1, \dots, n_k
- ▶ $(+\hat{\ } n_1 \dots n_k)$: All consequent formulas but n_1, \dots, n_k

Extended Expression Notation

- ▶ Term indexes:
 - ▶ L,R: Left- or right-hand side of a formula.
 - ▶ n: n-th term from left to right in a formula.
 - ▶ -n: n-th term from right to left in a formula.
 - ▶ *: All terms in a formula.
 - ▶ ($\hat{\ } n_1 \dots n_k$): All terms in a formula but n_1, \dots, n_k .
- ▶ Location references:
 - ▶ (! fnum LR $i_1 \dots i_k$): Term in formula fnum, Left- or Right-hand side, at recursive path location $i_1 \dots i_k$.

Simplifications via Field

- ▶ Field is a package for simplifications in the closed field of real numbers.
- ▶ <http://shemesh.larc.nasa.gov/people/cam/Field>.
- ▶ The package consists of:
 - ▶ The strategies `grind-reals` and `field`.
 - ▶ Several *extra-ategies*.
 - ▶ `Field` is pre-installed in PVS 5.0.

grind-reals

```
{-1} x * y >= 0
```

```
{-2} x > 0
```

```
|-----
```

```
{1} y >= 0
```

```
Rule? (grind-reals)
```

```
Rewriting with pos_times_ge
```

```
Applying GRIND-REALS,
```

```
Q.E.D.
```

field

$$\{-1\} \quad vox > 0$$

$$\{-2\} \quad s * s - D*D > D$$

$$\{-3\} \quad s * vix * voy - s * viy * vox \neq 0$$

$$\{-4\} \quad ((s * s - D*D) * voy - D * vox * \text{sqrt}(s*s - D*D)) / \\ (s * (vix * voy - vox * viy)) * s * vox \neq 0$$

$$\{-5\} \quad voy * \text{sqrt}(s * s - D*D) - D * vox \neq 0$$

|-----

$$\{1\} \quad (viy * \text{sqrt}(s * s - D*D) - vix * D) / \\ (voy * \text{sqrt}(s * s - D*D) - vox * D) = \\ (D*D - s * s) / (((s * s - D*D) * voy - D * vox * \\ \text{sqrt}(s * s - D*D)) / \\ (s * (vix * voy - vox * viy)) * s * vox) + \\ vix / vox$$

Rule? (field)

Q.E.D.

Some Extra-tergies

Strategy	Description
(cancel-by fnum term)	Cancel a common term in a formula
(skoletin fnum)	Skolemize let-in expressions
(skeep fnum)	Skolemize with same variable names
(neg-formula fnum)	Negate a formula
(add-formula fnum fnum)	Add two formulas

Interval Arithmetic via Interval

<http://shemesh.larc.nasa.gov/people/cam/Interval>

- ▶ Interval is a package for interval analysis.
- ▶ The package consists of:
 - ▶ The library `interval_arith`, which presents a formalization of interval analysis for real-valued functions including: trigonometric functions, logarithm and exponential functions, square root, absolute value, etc.
 - ▶ The strategy `numerical`, which implements a provably correct branch-and-bound interval analysis algorithm.
- ▶ Interval is part of the NASA PVS Libraries.

A Simple Problem

Prove that the turn rate of an aircraft with a bank angle of 35° is greater than 3° per second.

```
IMPORTING interval_arith@strategies
```

```
g:posreal=9.8          %[m/s^2]
```

```
v:posreal=250*0.514  %[m/s]
```

```
tr(phi:(Tan?)): MACRO real = g*tan(phi)/v
```

```
tr_35 : LEMMA
```

```
  3*pi/180 <= tr(35*pi/180)
```

numerical

tr_35 :

|-----
{1} 3 * pi / 180 <= g * tan(35 * pi / 180) / v

Rule? (numerical)

Evaluating formula using numerical approximations,
Q.E.D.

Special prize: Prove this lemma in any theorem prover different from PVS (Note: pi is the mathematical irrational number π and tan is the trigonometric function tan).

A Simple Property of Logarithms

```
G(x:real|x < 1): MACRO real = 3*x/2 - ln(1-x)
```

```
A_and_S : LEMMA
```

```
  let x = 0.5828 in
```

```
    G(x) > 0
```

A Simple Property of Logarithms

A_and_S :

```
|-----  
{1} LET x = 0.5828 IN 3 * x / 2 - ln(1 - x) > 0
```

Rule? (numerical)

Evaluating formula using numerical approximations,

Q.E.D.

Special prize: Prove this lemma in any theorem prover different from PVS (Note: \ln is natural logarithm function).

Interval Arithmetic

```
{-1} x ## [| 0, 2 |]
```

```
|-----
```

```
{1} sqrt(x) + sqrt(3) < pi + 0.1
```

Rule? (numerical :vars "x")

Evaluating formula using numerical approximations,

Q.E.D.

Interval Analysis

Prove that for all $x \in [-\frac{1}{2}, 0]$,

$$|\ln(1+x) - x| - \varepsilon \leq 2x^2,$$

where $\varepsilon = 0.15$:¹

```
ex_ba : LEMMA
```

```
  x ## [-1/2,0] IMPLIES
```

```
  abs(ln(1+x) - x) - epsilon <= 2*sq(x)
```

¹Thanks to Behzad Akbarpour.

instint

```

ex_ba :
  |-----
{1} FORALL (x: real):
      x ## [| -1/2, 0 |] IMPLIES abs(ln(1+x)-x)-0.15 <= 2*sq(x)

```

Rule? (skip)

```

ex_ba :
{-1} x ## [| -1 / 2, 0 |]
  |-----
{1} abs(ln(1 + x) - x) - 0.15 <= 2 * sq(x)

```

Rule? (numerical :vars (("x" 10)))

Evaluating formula using numerical approximations,
Q.E.D.

Solving Polynomial Inequalities via Bernstein

<http://shemesh.larc.nasa.gov/people/cam/Bernstein>

- ▶ Bernstein is a package for solving multivariate polynomial global optimization problems using Bernstein polynomials.
- ▶ The package consists of:
 - ▶ The library `Bernstein`, which presents a formalization of an efficient representation of multivariate polynomials.
 - ▶ The strategy `bernstein`, which discharges simply quantified multivariate polynomial inequalities on closed/open ranges.
 - ▶ `Grizzly`, which is a prototype client-server tool for solving global optimization problems.
- ▶ **Bernstein is part of the NASA PVS Libraries.**

Solving Polynomial Inequalities

```
IMPORTING Bernstein@strategy
```

```
p1 : LEMMA
```

```
  FORALL (x,y:real): -0.5 <= x AND x <= 1 AND
```

```
                    -2 <= y AND y <= 1 IMPLIES
```

```
                    4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 > -3.4
```

```
p2 : LEMMA
```

```
  EXISTS (x,y:real): -0.5 <= x AND x <= 1 AND
```

```
                    -2 <= y AND y <= 1 AND
```

```
                    4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 < -3.39
```

|-----
 {1} FORALL (x, y: real):
 -0.5 <= x AND x <= 1 AND -2 <= y AND y <= 1 IMPLIES
 4*x^2-(21/10)*x^4+(1/3)*x^6+(x-3)*y-4*y^2+4*y^4 > -3.4

Rule? (bernstein)

Proving polynomial inequality using Bernstein'basis,

Q.E.D.

Special prize: Prove this lemma in any theorem prover different from PVS

|-----

 $\{1\}$ EXISTS $(x, y: \text{real})$:

$$-0.5 \leq x \text{ AND } x \leq 1 \text{ AND } -2 \leq y \text{ AND } y \leq 1 \text{ AND}$$

$$4x^2 - (21/10)x^4 + (1/3)x^6 + (x-3)y - 4y^2 + 4y^4 < -3.39$$

Rule? (bernstein)

Proving polynomial inequality using Bernstein's basis,
Q.E.D.

Special prize: Prove this lemma in any theorem prover different
from PVS

A Final Example

```
cd2d_numeric_conflict: LEMMA
  -10 <= s'x AND s'x <= -8 AND
  -10 <= s'y AND s'y <= -8 AND
  6 <= v'x AND v'x <= 9 AND
  6 <= v'y AND v'y <= 9 AND
  D>=4 AND D<=6
  IMPLIES
  cd2d_ever?(s,v)
```

The proof is achieved with (grind), followed by (bernstein).

Concluding Remarks

Formal Methods in NextGen:

- ▶ NextGen is a system of systems: aircraft, physical environment, human operators.
- ▶ Formal methods for system engineering rather than for software engineering.
- ▶ Different sources of uncertainty.
- ▶ Highly distributed safety critical systems.

Practical Challenges

- ▶ Evolutionary vs. revolutionary concepts.
- ▶ Theoretical vs. practical solutions.
- ▶ Local vs. global solutions.

Current Technical Challenges

Automation, automation, automation:

- ▶ Non-linear arithmetic.
- ▶ Floating point arithmetic.
- ▶ Probabilistic reasoning.
- ▶ Numerical integration.