National Aeronautics and Space Administration

Applied Nonlinear Arithmetic in PVS

Anthony Narkawicz
NASA Langley Research Center

In collaboration with
César Muñoz
Aaron Dutle

Carnegie Mellon University
June 2018

Aeronautics

NASA Langley Research Center:

Approximately 1/3 Space

and 2/3 Aeronautics

Safety

Safety Critical Avionics Systems Branch

We design **high-integrity** prototype software for research into safety-critical aircraft systems

We don't want incorrect algorithms in safety-critical aircraft systems

Mathematics

Formal Methods team at NASA Langley

- 4 civil servants and 4 contracts using the **PVS theorem prover**
- The Prototype Verification System

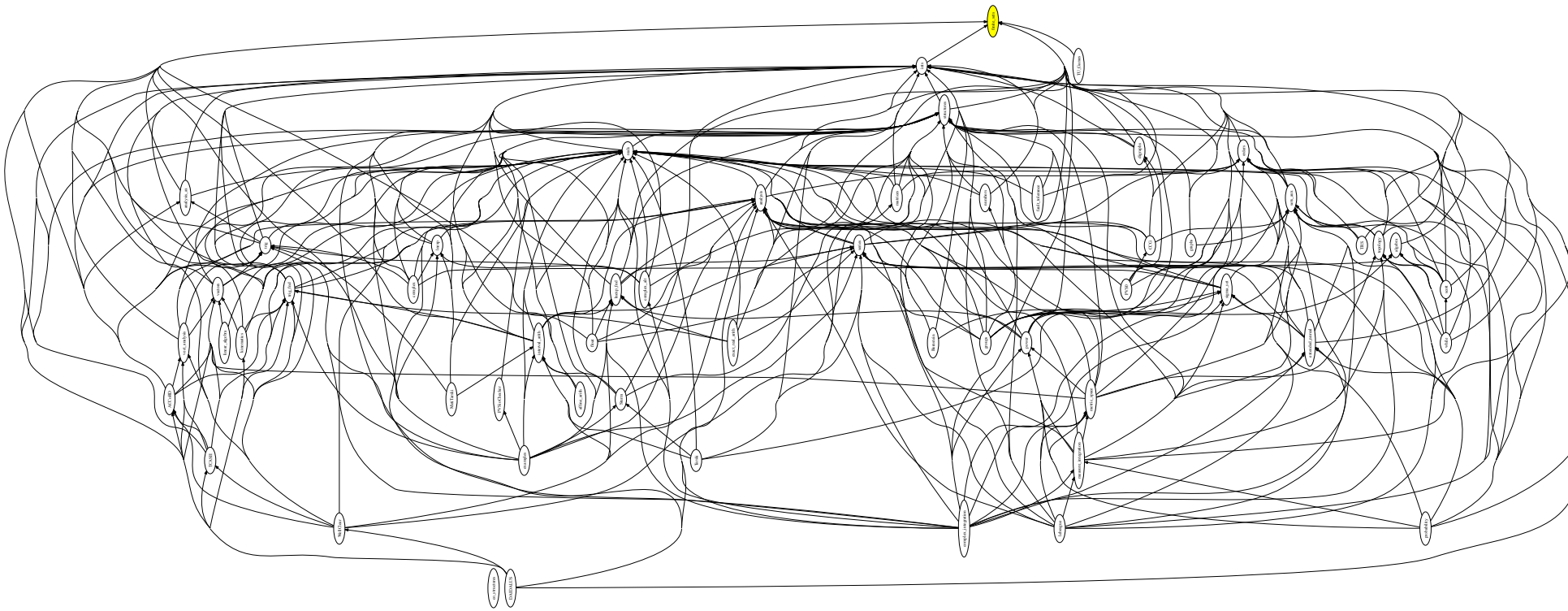
We have designed software to make aircraft avoid obstacles and other aircraft

Team has mathematicians and computer scientists

Theorem Proving at NASA

The formal methods group at NASA Langley has proved $> 20K$ theorems in PVS from a variety of areas of mathematics

NASA PVS Libraries: `https://github.com/nasa/pvslib`



The NASA PVS Libraries

PVS libraries in popular culture

`https://shemesh.larc.nasa.gov/people/cam/TheMartian/`

Example: Daidalus

We have used PVS to verify many of the algorithms in DAIDALUS:

DAIDALUS

Detect and AvoID Alerting Logic for Unmanned Systems

Example: Daidalus

DAIDALUS is the *reference implemenation* for detect and avoid for unmanned aircraft systems in the national airspace

DAIDALUS is part of the standards document RTCA DO-365

DAIDALUS

$\mathbf{s}_o, \mathbf{v}_o$	Horizontal component of ownship's position and velocity
$s_{oz}, \mathbf{v}_{oz}$	Ownship's altitude and vertical speed
$\mathbf{s}_i, \mathbf{v}_i$	Horizontal component of intruder's position and velocity
$s_{iz}, \mathbf{v}_{iz}$	Intruder's altitude and vertical speed

Let $\mathbf{s} = \mathbf{s}_o - \mathbf{s}_i$ and $\mathbf{v} = \mathbf{v}_o - \mathbf{v}_i$,

$$t_{\mathsf{cpa}}(\mathbf{s}, \mathbf{v}) \equiv -\frac{\mathbf{s} \cdot \mathbf{v}}{\mathbf{v}}, \qquad \tau(\mathbf{s}, \mathbf{v}) \equiv -\frac{\mathbf{s}^2}{\mathbf{s} \cdot \mathbf{v}}, \qquad \tau_{\mathsf{mod}}(\mathbf{s}, \mathbf{v}) \equiv \frac{\mathtt{DMOD}^2 - \mathbf{s}^2}{\mathbf{s} \cdot \mathbf{v}}.$$

DAIDALUS computes violations with aircraft based on the above functions

Definition of Well-Clear for Unmanned Aircraft

Aircraft have a violation of **well-clear** if they are inside the intersection of horizontal and vertical volumes

$$WCV(\mathbf{s}, \mathbf{s}_z, \mathbf{v}, \mathbf{v}_z) \equiv \texttt{Horizontal_WCV}(\mathbf{s}, \mathbf{v}) \ \texttt{and} \ \texttt{Vertical_WCV}(\mathbf{s}_z, \mathbf{v}_z), \quad (1)$$

Inside this volume the aircraft is **not well clear**.

$$\begin{aligned}
\texttt{Horizontal_WCV}(\mathbf{s}, \mathbf{v}) &\equiv \|\mathbf{s}\| \leq \texttt{DMOD} \ \texttt{or} \\
&\quad (d_{\mathsf{cpa}}(\mathbf{s}, \mathbf{v}) \leq \texttt{DMOD} \ \texttt{and} \ 0 \leq \tau_{\mathsf{mod}}(\mathbf{s}, \mathbf{v}) \leq \texttt{TAUMOD}), \\
\texttt{Vertical_WCV}(\mathbf{s}_z, \mathbf{v}_z) &\equiv |\mathbf{s}_z| \leq \texttt{ZTHR} \ \texttt{or} \ 0 \leq t_{\mathsf{coa}}(\mathbf{s}_z, \mathbf{v}_z) \leq \texttt{TCOA}.
\end{aligned}$$

Daidalus v2.0

Our current project is Daidalus v2.0

RTCA standards organization requested that we incorporate sensor uncertainty into DAIDALUS

GPS, Radar, and other sensors give information such as variances and covariances for East/North/Up-Down components of positions and velocities

We are trying several different approaches to this problem

Daidalus v2.0

Positions and velocities are random variables

- Inputs $\mathbf{s}, \mathbf{v}$ (reported relative position/velocity) and X,Y (Random Variables)
- Horizontal position is $\mathbf{s} + \begin{pmatrix} X \\ Y \end{pmatrix}$
- Horizontal velocity is $\mathbf{v} + M \cdot \begin{pmatrix} X \\ Y \end{pmatrix}$ (M is a rotation matrix)

We compute random variables, variances, covariances, means, for functions in the well-clear formula

$$[\|\mathbf{s}\| \leq \texttt{DMOD} \ \texttt{or} \ (d_{\text{cpa}}(\mathbf{s}, \mathbf{v}) \leq \texttt{DMOD} \ \texttt{and} \ 0 \leq \tau_{\text{mod}}(\mathbf{s}, \mathbf{v}) \leq \texttt{TAUMOD})]$$
$$\texttt{and} \ [|\mathbf{s}_z| \leq \texttt{ZTHR} \ \texttt{or} \ 0 \leq t_{\text{coa}}(\mathbf{s}_z, \mathbf{v}_z) \leq \texttt{TCOA}]$$

Daidalus v2.0

The random variable for the d_{cpa} function at a future time t is

```
sqv(s)+sqv(v)*t^2+(s*v)*TAUMOD+2*((s*v)*t)-
sq(DMOD)+(v*v)*TAUMOD*t+((2*s`x+v`x*TAUMOD+2*(v`x*t)+s`x*TAUMOD*a+
2*(s`x*a*t)+s`y*TAUMOD*c+2*(s`y*c*t)+2*(v`x*t^2*a)+2*(v`y*t^2*c)
+2*(v`x*TAUMOD*a*t)+2*(v`y*TAUMOD*c*t))*X)+((2*s`y+v`y*TAUMOD+2*(v`y*t)+s`x*TAUMOD*b+
2*(s`x*b*t)+s`y*TAUMOD*d+2*(s`y*d*t)+2*(v`x*t^2*b)+2*(v`y*t^2*d)+2*(v`x*TAUMOD*b*t)
+2*(v`y*TAUMOD*d*t))*Y)+(1+sq(a)*t^2+sq(c)*t^2+TAUMOD*a+sq(a)*TAUMOD*t+sq(c)*TAUMOD*t
+2*(a*t))*(X*X)+(1+sq(b)*t^2+sq(d)*t^2+TAUMOD*d+sq(b)*TAUMOD*t+sq(d)*TAUMOD*t+2*(d*t))*(Y*Y)
+(TAUMOD*b+TAUMOD*c+2*(t^2*a*b)+2*(t^2*c*d)+2*(TAUMOD*a*b*t)+2*(TAUMOD*c*d*t)+2*(b*t)+2*(c*t))*(X*Y)
```

Daidalus v2.0

It's variance is given by

```
real = 2*(sq(VX)*sq(sq(s`x)*sq(c)+sq(s`y)*sq(a)-2*(s`x*s`y*a*c)-sq(DMOD)*sq(a)-sq(DMOD)*sq(c)))
+2*(sq(VY)*sq(sq(s`x)*sq(d)+sq(s`y)*sq(b)-2*(s`x*s`y*b*d)-sq(DMOD)*sq(b)-sq(DMOD)*sq(d)))
+sq(CXY)*sq(-2*(sq(DMOD)*a*b)-2*(sq(DMOD)*c*d)-2*(s`x*s`y*a*d)-2*(s`x*s`y*b*c)+2*(sq(s`x)*c*d)+2*(sq(s`y)*a*b))
+sq(-2*(s`y*det(s,v)*a)-2*(sq(DMOD)*v`x*a)-2*(sq(DMOD)*v`y*c)+2*(s`x*det(s,v)*c))*VX
+sq(-2*(s`y*det(s,v)*b)-2*(sq(DMOD)*v`x*b)-2*(sq(DMOD)*v`y*d)+2*(s`x*det(s,v)*d))*VY-16*
(VX*s`x*s`y*sq(s`x)*CXY*a*c*c*d)-16*(VX*s`x*s`y*sq(s`y)*CXY*a*a*b*c)-16*(VY*s`x*s`y*sq(s`x)*CXY*b*c*d*d)-16*
(VY*s`x*s`y*sq(s`y)*CXY*a*b*b*d)-8*(VX*sq(s`x)*sq(DMOD)*sq(a)*CXY*c*d)-8*
(VX*sq(s`x)*sq(DMOD)*sq(c)*CXY*a*b)-8*(VX*sq(s`x)*sq(DMOD)*sq(c)*CXY*c*d)-8*
(VX*sq(s`x)*sq(DMOD)*sq(c)*CXY*c*d)-8*(VX*sq(s`y)*sq(DMOD)*sq(a)*CXY*a*b)-8*
(VX*sq(s`y)*sq(DMOD)*sq(a)*CXY*a*b)-8*(VX*sq(s`y)*sq(DMOD)*sq(a)*CXY*c*d)-8*
(VX*sq(s`y)*sq(DMOD)*sq(c)*CXY*a*b)-8*(VY*sq(s`x)*sq(DMOD)*sq(b)*CXY*c*d)-8*
(VY*sq(s`x)*sq(DMOD)*sq(d)*CXY*a*b)-8*(VY*sq(s`x)*sq(DMOD)*sq(d)*CXY*c*d)-8*
(VY*sq(s`x)*sq(DMOD)*sq(d)*CXY*c*d)-8*(VY*sq(s`y)*sq(DMOD)*sq(b)*CXY*a*b)-8*
(VY*sq(s`y)*sq(DMOD)*sq(b)*CXY*a*b)-8*(VY*sq(s`y)*sq(DMOD)*sq(b)*CXY*c*d)-8*
(VY*sq(s`y)*sq(DMOD)*sq(d)*CXY*a*b)-8*(s`x*s`y*sq(s`x)*sq(CXY)*sq(c)*b*d)
-8*(s`x*s`y*sq(s`x)*sq(CXY)*sq(d)*a*c)-8*(s`x*s`y*sq(s`y)*sq(CXY)*sq(a)*b*d)
-8*(s`x*s`y*sq(s`y)*sq(CXY)*sq(b)*a*c)-8*(s`x*s`y*CXY*det(s,v)*det(s,v)*a*d)
-8*(s`x*s`y*CXY*det(s,v)*det(s,v)*b*c)-8*(s`x*sq(DMOD)*v`x*CXY*det(s,v)*a*d)
-8*(s`x*sq(DMOD)*v`x*CXY*det(s,v)*b*c)-8*(s`x*sq(DMOD)*v`y*CXY*det(s,v)*c*d)
-8*(s`x*sq(DMOD)*v`y*CXY*det(s,v)*c*d)-8*(VX*s`x*s`y*sq(s`x)*sq(c)*CXY*a*d)
-8*(VX*s`x*s`y*sq(s`x)*sq(c)*CXY*b*c)-8*(VX*s`x*s`y*sq(s`y)*sq(a)*CXY*a*d)-8*
(VX*s`x*s`y*sq(s`y)*sq(a)*CXY*b*c)-8*(VY*s`x*s`y*sq(s`x)*sq(d)*CXY*a*d)-8*
(VY*s`x*s`y*sq(s`x)*sq(d)*CXY*b*c)-8*(VY*s`x*s`y*sq(s`y)*sq(b)*CXY*a*d)-8*
(VY*s`x*s`y*sq(s`y)*sq(b)*CXY*b*c)-4*(sq(s`x)*sq(CXY)*sq(DMOD)*sq(a)*sq(d))-4*(sq(s`x)*sq(CXY)*sq(DMOD)*sq(b)*sq(c))
-4*(sq(s`x)*sq(CXY)*sq(DMOD)*sq(c)*sq(d))-4*(sq(s`x)*sq(CXY)*sq(DMOD)*sq(c)*sq(d))
-4*(sq(s`y)*sq(CXY)*sq(DMOD)*sq(a)*sq(b))-4*(sq(s`y)*sq(CXY)*sq(DMOD)*sq(a)*sq(b))
-4*(sq(s`y)*sq(CXY)*sq(DMOD)*sq(a)*sq(d))-4*(sq(s`y)*sq(CXY)*sq(DMOD)*sq(b)*sq(c))
+VX*VY*sq(-2*(sq(DMOD)*a*b)-2*(sq(DMOD)*c*d)-2*(s`x*s`y*a*d)-2*(s`x*s`y*b*c)+2*(sq(s`x)*c*d)
+2*(sq(s`y)*a*b))+4*(sq(s`x)*sq(s`x)*sq(CXY)*sq(c)*sq(d))
+4*(sq(s`x)*sq(s`y)*sq(CXY)*sq(a)*sq(d))+4*(sq(s`x)*sq(s`y)*sq(CXY)*sq(b)*sq(c))
+4*(sq(s`y)*sq(s`y)*sq(CXY)*sq(a)*sq(b))+4*(sq(CXY)*sq(DMOD)*sq(DMOD)*sq(a)*sq(b))
+4*(sq(CXY)*sq(DMOD)*sq(DMOD)*sq(a)*sq(d))+4*(sq(CXY)*sq(DMOD)*sq(DMOD)*sq(b)*sq(c))
+4*(sq(CXY)*sq(DMOD)*sq(DMOD)*sq(c)*sq(d))+8*(VX*sq(s`x)*sq(s`x)*sq(c)*CXY*c*d)+
8*(VX*sq(s`x)*sq(s`y)*sq(a)*CXY*c*d)+8*(VX*sq(s`x)*sq(s`y)*sq(c)*CXY*a*b)+8*
(VX*sq(s`y)*sq(s`y)*sq(a)*CXY*a*b)+8*(VX*sq(DMOD)*sq(DMOD)*sq(a)*CXY*a*b)+8*
(VX*sq(DMOD)*sq(DMOD)*sq(a)*CXY*c*d)+8*(VX*sq(DMOD)*sq(DMOD)*sq(c)*CXY*a*b)+8*
(VX*sq(DMOD)*sq(DMOD)*sq(c)*CXY*c*d)+8*(VY*sq(s`x)*sq(s`x)*sq(d)*CXY*c*d)+8*
(VY*sq(s`x)*sq(s`y)*sq(b)*CXY*c*d)+8*(VY*sq(s`x)*sq(s`y)*sq(d)*CXY*a*b)+8*
(VY*sq(s`y)*sq(s`y)*sq(b)*CXY*a*b)+8*(VY*sq(DMOD)*sq(DMOD)*sq(b)*CXY*a*b)+
8*(VY*sq(DMOD)*sq(DMOD)*sq(b)*CXY*c*d)+8*(VY*sq(DMOD)*sq(DMOD)*sq(d)*CXY*a*b)
+8*(VY*sq(DMOD)*sq(DMOD)*sq(d)*CXY*c*d)+8*(sq(s`x)*CXY*det(s,v)*det(s,v)*c*d)
+8*(s`x*s`y*sq(CXY)*sq(DMOD)*sq(a)*b*d)+8*(s`x*s`y*sq(CXY)*sq(DMOD)*sq(b)*a*c)
+8*(s`x*s`y*sq(CXY)*sq(DMOD)*sq(c)*b*d)+8*(s`x*s`y*sq(CXY)*sq(DMOD)*sq(d)*a*c)
+8*(sq(s`y)*CXY*det(s,v)*det(s,v)*a*b)+8*(s`y*sq(DMOD)*v`x*CXY*det(s,v)*a*b)
+8*(s`y*sq(DMOD)*v`x*CXY*det(s,v)*a*b)+8*(s`y*sq(DMOD)*v`y*CXY*det(s,v)*a*d)
+8*(s`y*sq(DMOD)*v`y*CXY*det(s,v)*b*c)+8*(sq(DMOD)*sq(DMOD)*sq(v`x)*CXY*a*b)
+8*(sq(DMOD)*sq(DMOD)*v`x*v`y*CXY*a*d)+8*(sq(DMOD)*sq(DMOD)*v`x*v`y*CXY*b*c)
+8*(sq(DMOD)*sq(DMOD)*sq(v`y)*CXY*c*d)+8*(VX*s`x*s`y*sq(DMOD)*sq(a)*CXY*a*d)+8*
(VX*s`x*s`y*sq(DMOD)*sq(a)*CXY*b*c)+8*(VX*s`x*s`y*sq(DMOD)*sq(c)*CXY*a*d)+8*
(VX*s`x*s`y*sq(DMOD)*sq(c)*CXY*b*c)+8*(VY*s`x*s`y*sq(DMOD)*sq(b)*CXY*a*d)+8*
(VY*s`x*s`y*sq(DMOD)*sq(b)*CXY*b*c)+8*(VY*s`x*s`y*sq(DMOD)*sq(d)*CXY*a*d)+8*
(VY*s`x*s`y*sq(DMOD)*sq(d)*CXY*b*c)+16*(VX*s`x*s`y*sq(DMOD)*CXY*a*a*b*c)+16*
(VX*s`x*s`y*sq(DMOD)*CXY*a*c*c*d)+16*(VY*s`x*s`y*sq(DMOD)*CXY*a*b*b*d)+
16*(VY*s`x*s`y*sq(DMOD)*CXY*b*c*d*d)+16*(sq(s`x)*sq(s`y)*sq(CXY)*a*b*c*d)+
16*(VX*sq(s`x)*sq(s`y)*CXY*a*a*c*d)+16*(VX*sq(s`x)*sq(s`y)*CXY*a*b*c*c)+
16*(VY*sq(s`x)*sq(s`y)*CXY*a*b*d*d)+16*(VY*sq(s`x)*sq(s`y)*CXY*b*b*c*d)
```

This is formally proved in PVS.

The proofs involve reasoning about large and complex expressions with real numbers.

Daidalus v2.0

In the verification, we have to prove

```
Delta_sum: LEMMA sqv(s)>sq(D) AND s*v<0 AND s*nv<0 AND
  Delta[D](s,v)>=0 AND Delta[D](s,nv)>=0 IMPLIES
  Delta[D](s,v+nv)>=0
```

where

```
Delta(s,v) : real =
  sq(D)*sqv(v) - sq(det(s,v))
```

Daidalus v2.0

The proof in PVS reduces to this:

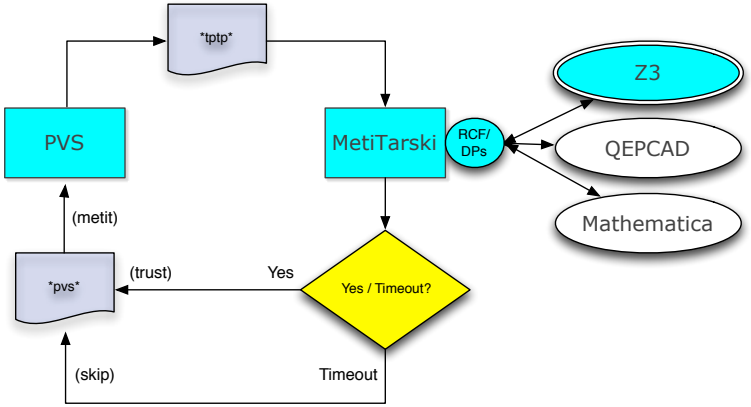
```
[-1]  s`x * s`x + s`y * s`y > D * D
[-2]  s`x * v`x + s`y * v`y < 0
[-3]  s`x * nv`x + s`y * nv`y < 0
[-4]  -1 * (s`x * s`x * v`y * v`y) - s`y * s`y * v`x * v`x +
       2 * (s`x * s`y * v`x * v`y)
       + v`x * v`x * D * D
       + v`y * v`y * D * D
       >= 0
[-5]  -1 * (nv`x * nv`x * s`y * s`y) - nv`y * nv`y * s`x * s`x +
       nv`x * nv`x * D * D
       + 2 * (nv`x * nv`y * s`x * s`y)
       + nv`y * nv`y * D * D
       >= 0
  |-------
{1}   -2 * (nv`x * s`y * s`y * v`x) - 2 * (nv`y * s`x * s`x * v`y) -
       nv`x * nv`x * s`y * s`y
       - nv`y * nv`y * s`x * s`x
       - s`x * s`x * v`y * v`y
       - s`y * s`y * v`x * v`x
       + ((nv`x + v`x) * (nv`x + v`x)) * D * D
       + ((nv`y + v`y) * (nv`y + v`y)) * D * D
       + 2 * (nv`x * nv`y * s`x * s`y)
       + 2 * (nv`x * s`x * s`y * v`y)
       + 2 * (nv`y * s`x * s`y * v`x)
       + 2 * (s`x * s`y * v`x * v`y)
       >= 0

Rule?
```

How do we prove this?

Daidalus v2.0

```
Rule? (metit *)
Substitutions: D -> V1, nv`x -> V2, nv`y -> V3, s`x -> V4, s`y -> V5, v`x -> V6, v`y -> V7
MetiTarski Input =
 fof(pvs2metit,conjecture, (![V1, V2, V3, V4, V5, V6, V7]: ((~ (((V4 * V4) + (V5 * V5)) > (V1 * V1))) | ((~ (((V4 * V6) + (V5
((V4 * V2) + (V5 * V3)) < 0)) | ((~ (((((((-1 / 1) * (((V4 * V4) * V7) * V7)) - (((V5 * V5) * V6) * V6)) + (2 * (((V4 * V5) *
* V6) * V1) * V1)) + (((V7 * V7) * V1) * V1)) >= 0)) | ((~ (((((((-1 / 1) * (((V2 * V2) * V5) * V5)) - (((V3 * V3) * V4) * V4
) * V1)) + (2 * (((V2 * V3) * V4) * V5))) + (((V3 * V3) * V1) * V1)) >= 0)) | (((((((((((((-2 / 1) * (((V2 * V5) * V5) * V6)
* V4) * V7))) - (((V2 * V2) * V5) * V5)) - (((V3 * V3) * V4) * V4)) - (((V4 * V4) * V7) * V7)) - (((V5 * V5) * V6) * V6)) + (
6)) * V1) * V1)) + ((((V3 + V7) * (V3 + V7)) * V1) * V1)) + (2 * (((V2 * V3) * V4) * V5))) + (2 * (((V2 * V4) * V5) * V7))) +
5) * V6))) + (2 * (((V4 * V5) * V6) * V7))) >= 0)))))))).

Result = Generated Include List
Axioms/trans.ax
Axioms/general.ax
Axioms/minmax.ax
----------------------------
SZS status Theorem for /Users/anarkawi/Documents/SVN_Software/DAIDALUS/SUM/optimal_vectors/pvsbin/Delta_sum.tptp
Processor time: 0.782 = 0.058 (Metis) + 0.724 (RCF)
Maximum weight in proof search: 17
MetiTarski succesfully proved.
Trusted oracle: MetiTarski.

Proving formula(s) * with MetiTarski,
Q.E.D.



Run time  = 1.93 secs.
Real time = 183.42 secs.

nil
pvs(18):
```

Integration of MetiTarski into PVS



This calls MetiTarski/Z3, which uses the algorithm from the book

Algorithms in Real Algebraic Geometry by Basu, Pollack, and Roy

In Z3: Dejan Jovanovic and Leonardo de Moura. Solving non-linear arithmetic. In Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings, volume 7364 of Lecture Notes in Computer Science, pages 339-354. Springer, 2012.

Integration of MetiTarski into PVS

Formally proving the algorithm in PVS:

- Metit is used as an oracle in PVS (an addition to the kernel)
- Ultimate Goal: Build and formally prove a version of metit on *top* of the kernel of PVS
- We have two different implementations for the univariate case
- ... and **zero** implementations for the multivariate case

Exact Real Algebraic Geometry

Exact Real Algebraic Geometry

- Sturm Sequences
- Tarski Queries

Based on exact algorithms in algebraic geometry.

More complicated mathematics

Tarski queries use Sturm sequences

Tarski queries determine if any polynomial system
$p_1(x_1, \ldots, x_n) > 0 \wedge \cdots \wedge p_m(x_1, \ldots, x_m) \leq 0$ has a solution.

Tarski's Theorem

Let p and g be polynomials. Compute the remainder sequence

$$p_0(x),\ p_1(x),\ \ldots,\ p_m(x),$$

where $p_0 = p$, $p_1 = g \cdot p'$, $p_{j+2} = -rem(p_j, p_{j+1})$, and $p_m \equiv 0$

$\mathtt{TQ}(p, q)$ - A Tarski Query - a computable function based on the sequence $p_0, \ldots, p_m$.

<u>Tarski's Basic Theorem</u>

$$\mathtt{card}\{x \mid p(x) = 0 \wedge g(x) > 0\} - \mathtt{card}\{x \mid p(x) = 0 \wedge g(x) < 0\} = \mathtt{TQ}(p, g)$$

Tarski's Theorem

Book: *Algorithms in Real Algebraic Geometry* by Saugata Basu, Richard Pollack, Marie-Franoise Roy

Tarski's (Basic) Matrix Theorem

$$\begin{bmatrix} \mathrm{TQ}(p,1) \\ \mathrm{TQ}(p,g) \\ \mathrm{TQ}(p,g^2) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & -1 \\ 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)=0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)>0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)<0\}) \end{bmatrix}$$

Corollary

$$\begin{bmatrix} \mathrm{TQ}(p,1) \\ \mathrm{TQ}(p,g) \\ \mathrm{TQ}(p,g^2) \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)=0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)>0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)<0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)\neq 0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)\geq 0\}) \\ \mathrm{card}(\{x \,:\, p(x)=0 \,\wedge\, g(x)\leq 0\}) \end{bmatrix}$$

Call this matrix $\mathtt{M}$

Multiple Polynomials

This is the base case of an induction proof for multiple polys

$$g_0, \ldots, g_k$$

There are two 6^{k+1} dim vectors $\mathtt{N}(p, g_0, \ldots, g_k)$ and $\mathtt{TQ}(p, g_0, \ldots, g_k)$:

- The entries of $\mathtt{N}(p, g_0, \ldots, g_k)$ are all the possibilities of

$$\mathtt{card}\{x \in \mathbb{R} : \; p(x) = 0 \;\wedge\; g_0(x)\; R_0\; 0 \;\;\wedge\;\; \ldots \;\;\wedge\;\; g_k(x)\; R_k\; 0\}$$

  where each $R_d \in \{=, >, <, \neq, \geq, \leq\}$

- The entries of $\mathtt{TQ}(p, g_0, \ldots, g_k)$ are Tarski queries (computable)

We just need to compute the entries of $\mathtt{N}(p, g_0, \ldots, g_k)$.

Tarski's Big Theorem

<u>Tarski</u>

$$\mathtt{TQ}(p, g_0, \ldots, g_k) = \mathtt{M}^{\otimes(k+1)} \cdot \mathtt{N}(p, g_0, \ldots, g_k)$$

<u>Corollary</u>

$$\mathtt{N}(p, g_0, \ldots, g_k) = (\mathtt{M}^{\otimes(k+1)})^{-1}\mathtt{TQ}(p, g_0, \ldots, g_k)$$

We want to compute entries of $\mathtt{N}(p, g_0, \ldots, g_k)$ (cards of solution sets)

Just need to compute $(\mathtt{M}^{\otimes(k+1)})^{-1} = (\mathtt{M}^{-1})^{\otimes(k+1)} =$

$$\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & -\frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & \frac{1}{2} & -\frac{1}{2} & 0 & 1 & 0 \\ 1 & -\frac{1}{2} & -\frac{1}{2} & 0 & 0 & 1 \end{bmatrix}^{\otimes(k+1)}$$

Sturm's Theorem

We can therefore compute cardinalities

$$\texttt{card}\{x \in \mathbb{R} : \; p(x) = 0 \;\wedge\; g_0(x) \; R_0 \; 0 \;\wedge\; \ldots \;\wedge\; g_k(x) \; R_k \; 0\}$$

We can also determine if

$$\texttt{card}\{x \in \mathbb{R} : \; g_0(x) \; R_0 \; 0 \;\wedge\; \ldots \;\wedge\; g_k(x) \; R_k \; 0\} \; = \; 0$$

By testing whether one of the following holds

- The first cardinality is zero with $p = \prod_i g_i$
- The first cardinality is zero with $p = \frac{d}{dx}(\prod_i g_i)$
- Either $-\infty$ or $+\infty$ is *in* this second set.

Decision Procedure

We can therefore decide if these cardinalities are zero:

$$\texttt{card}\{x \in \mathbb{R} : \; g_0(x) \; R_0 \; 0 \;\; \wedge \;\; \ldots \;\; \wedge \;\; g_k(x) \; R_k \; 0\} \; = \; 0$$

(i.e. If the system $g_0(x) \; R_0 \; 0 \;\; \wedge \;\; \ldots \;\; \wedge \;\; g_k(x) \; R_k \; 0$ has a solution)

```
example_1: LEMMA
  FORALL(y,x,z:real): (x-2)^2*(-x+4)>0 AND x^2*(x-3)^2>=0 AND x-1>=0 AND -(x-3)^2+1>0
  IMPLIES
  -(x-11/12)^3*(x-41/10)^3>=0
```

This proves now by simply typing "(tarski)" in the PVS prover. It's all automated.

Implementation of Tarski

- There is a deep embedding of the algorithm in PVS...
- ... and an theorem stating its correctness
- A property is proved by invoking the theorem in the prover...
- ... and the *evaluating* the resulting call to the algorithm...
- through reflection... The evaluation is done in PVS's underlying LISP language.

The PVS Strategy Tarski

A theorem in the PVS prover language:

$$\{-1\}\ x < 3$$
$$\vdash$$
$$\{\ 1\ \}\ x^{120} - \tfrac{2}{3}x^{60} + \tfrac{1}{9} \geq 0$$

Call the lemma:

$$\{-1\}\ \texttt{sturm}(p, I, \geq) \iff (\forall (x\colon \mathbb{R})\colon x \in I \implies p(x) \geq 0)$$
$$\{-2\}\ x < 3$$
$$\vdash$$
$$\{\ 1\ \}\ x^{120} - \tfrac{2}{3}x^{60} + \tfrac{1}{9} \geq 0$$

The PVS Strategy Tarski

Evaluate the function:

$$\{-1\}\ \texttt{true} \iff (\forall(x\colon \mathbb{R})\colon x \in I \implies p(x) \geq 0)$$
$$\{-2\}\ x < 3$$
$$\vdash$$
$$\{\ 1\ \}\ x^{120} - \tfrac{2}{3}x^{60} + \tfrac{1}{9} \geq 0$$

Simplify:

$$\{-1\}\ \forall(x\colon \mathbb{R})\colon x \in I \implies p(x) \geq 0$$
$$\{-2\}\ x < 3$$
$$\vdash$$
$$\{\ 1\ \}\ x^{120} - \tfrac{2}{3}x^{60} + \tfrac{1}{9} \geq 0$$

The PVS Strategy Tarski

Instantiate with x:

$\{-1\}\ x \in I \implies p(x) \geq 0$
$\{-2\}\ x < 3$
$\vdash$
$\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$

Split formula -1 into two cases:

$\{-1\}\ x < 3$
$\vdash$
$\{\ 1\ \}\ x \in I$
$\{\ 2\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$

$\{-1\}\ p(x) \geq 0$
$\{-2\}\ x < 3$
$\vdash$
$\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$

The PVS Strategy Tarski

$\{-1\}\ p(x) \geq 0$
$\{-2\}\ x < 3$
$\vdash$
$\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$

- $p(x)$ in formula -1 is a *deep embedding* of the polynomial
- $x^{120} - \frac{2}{3}x^{60} + \frac{1}{9}$ is a *real number*
- For some simple problems, this equivalence is easily proved by expanding definitions
- For others, that takes way too long
  - Automatically expanding ˆproduces 120 proof steps
  - Sometimes this real number expression is not yet simplified

The PVS Strategy Tarski

$\{-1\}\ p(x) \geq 0$
$\{-2\}\ x < 3$
$\vdash$
$\{\ 1\ \}\ x^{120} - \frac{2}{3}x^{60} + \frac{1}{9} \geq 0$

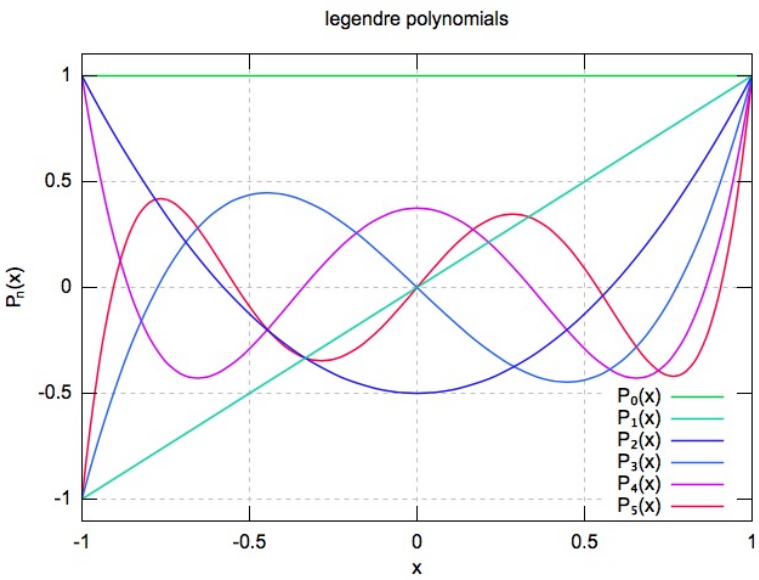
We prove this by rewriting using a set of rules:

- $\mathtt{polylist}(\mathtt{pconst}(c))(x) = c.$
- $\mathtt{polylist}(\mathtt{pmonom}(c, i))(x) = c\ x^i.$
- $\mathtt{polylist}(\mathtt{psum}(p, q))(x) = p(x) + q(x).$
- $\mathtt{polylist}(\mathtt{pneg}(p))(x) = -p(x).$
- $\mathtt{polylist}(\mathtt{pminus}(p, q))(x) = p(x) - q(x).$
- $\mathtt{polylist}(\mathtt{pprod}(p, q))(x) = p(x) \cdot q(x).$
- $\mathtt{polylist}(\mathtt{pscal}(c, p))(x) = c\ p(x).$
- $\mathtt{polylist}(\mathtt{pdiv}(p, c))(x) = \frac{p(x)}{c},\ when\ c \neq 0.$
- $\mathtt{polylist}(\mathtt{ppow}(p, i))(x) = p(x)^i.$

This can be a *little bit* slow if the sequent is large.

Hutch

A simpler method for proving unsatisfiability of a univariate system
$p_1(x) < 0 \wedge \cdots \wedge p_5(x) < 0$

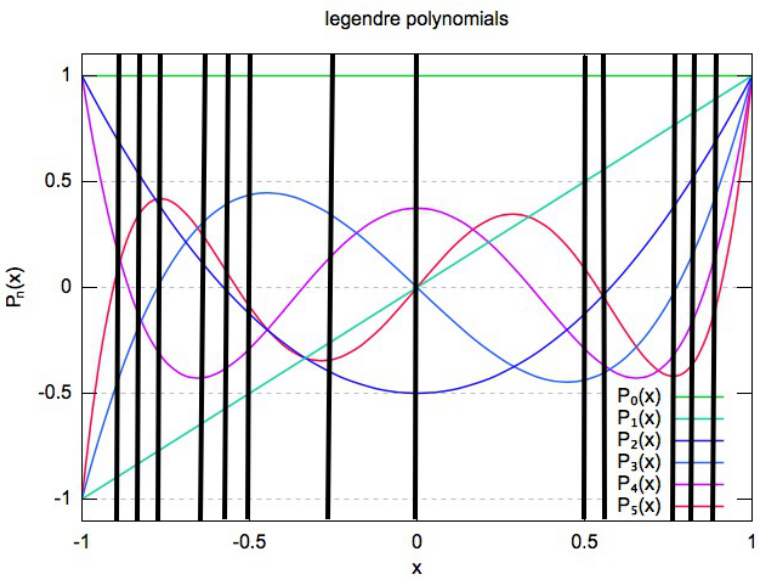


https://commons.wikimedia.org/wiki/File:Legendrepolynomials6.svg, edited: CC-by-SA 3.0,
https://creativecommons.org/licenses/by-sa/3.0/

- Subdivide until every every interval has at most one root of any polynomial.
- Check this by counting roots of $\sum_i p_i(x)^2$ or $\prod_i p_i(x)$...
- using Tarski queries.

Hutch

A simpler method for proving unsatisfiability of a univariate system
$p_1(x) < 0 \wedge \cdots \wedge p_5(x) < 0$



https://commons.wikimedia.org/wiki/File:Legendrepolynomials6.svg, edited: CC-by-SA 3.0,
https://creativecommons.org/licenses/by-sa/3.0/

- Subdivide until every every interval has at most one root of any polynomial.
- Check this by counting roots of $\sum_i p_i(x)^2$ or $\prod_i p_i(x)$...
- using Tarski queries.

Hutch

```
example_1: LEMMA
  FORALL(y,x,z:real): (x-2)^2*(-x+4)>0 AND x^2*(x-3)^2>=0 AND x-1>=0 AND -(x-3)^2+1>0
  IMPLIES
  -(x-11/12)^3*(x-41/10)^3>=0
```

This new command is called Hutch.

This proves now by simply typing "(tarski)" or "(hutch)" in the PVS prover. It's all automated.

The implemenations are similar and both use reflection.

Narkawicz, A. J.; Munoz, C. A.; and Dutle, A. M.: A Decision Procedure for Univariate Polynomial Systems Based on Root Counting and Interval Subdivision, Journal of Formalized Reasoning, 2018 (To appear)

Comparison of Tarski and Hutch

Problem	tarski	hutch	hutch :sos? nil	metit
Ex1	1.78 (0.12)	2.68 (0.02)	1.73 (0.02)	0.05
Ex2	4.80 (2.16)	2.91 (0.07)	3.19 (0.38)	0.05
Ex3	5.07 (0.26)	30.19 (27.22)	6.41 (1.74)	N/A
Ex4	12.58 (9.69)	4.07 (0.01)	4.09 (0.05)	0.04
Ex5	225.45 (237.96)	5.55 (0.01)	4.44 (0.17)	0.05
Ex6	–	70.02 (3.02)	–	N/A
Ex7	–	76.76 (51.85)	–	0.05
quad2	1.82 (0.01)	1.85 (0.00)	1.85 (0.00)	0.05
quad3	2.28 (0.05)	1.05 (0.00)	2.27 (0.00)	0.05
quad4	1.83 (0.44)	2.74 (0.00)	2.75 (0.01)	0.05
quad5	5.57 (2.88)	3.20 (0.01)	3.25 (0.03)	0.05
quad6	22.16 (21.61)	3.75 (0.01)	3.82 (0.08)	0.05
quad7	154.43 (175.47)	4.26 (0.01)	4.48 (0.24)	0.05
quad8	–	8.73 (0.01)	4.11 (0.53)	0.05
quad9	–	11.90 (0.01)	4.46 (1.09)	0.05
quad10	–	14.19 (0.02)	7.98 (2.07)	0.05

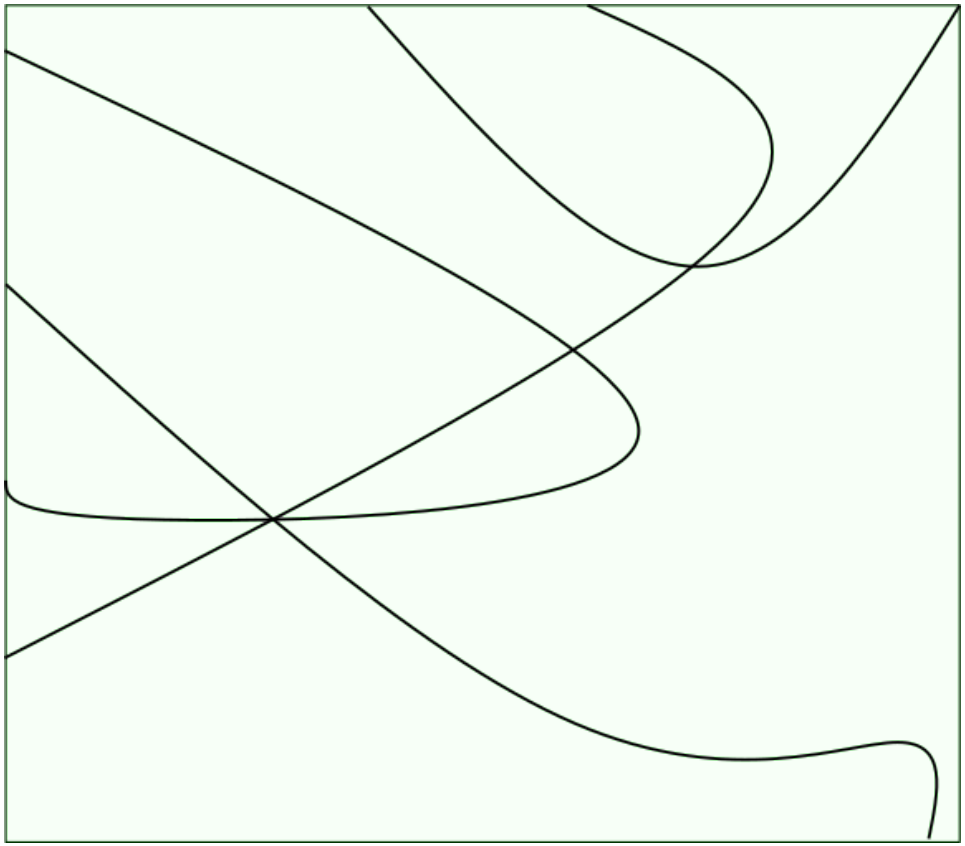
Multivariate Subdivision Algorithm

Is there a multivariate version of Hutch, and would that be the same as standard CAD?

- This would be much easier to prove in PVS
  - ... and add on top of the kernel (in a sound way)
- Subdivide n-dim box until every sub-box is small enough so that
  - For every nonempty subvariety of the form $\{x \mid \sum_i p_i(x)^2 = 0\}$ taken over a subset of the polynomials, either
    - it has positive dimension and is sliced by the edge of some box, *or*
    - it has zero dimension and is contained in a sub-box that contains no other zero dimensional subvariety
- Check satisfiability on the boundary of each sub-box **and** at every zero dimensional subvariety

Multivariate Subdivision Algorithm

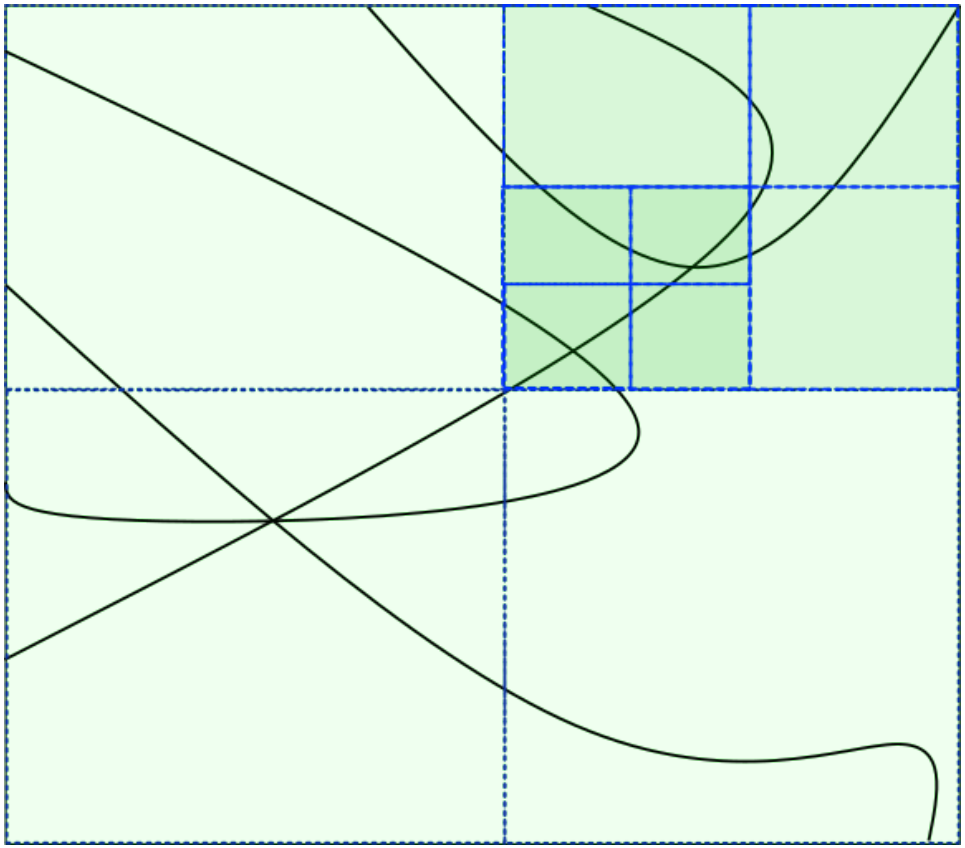
2-dimensional example: plot the zero sets of polynomials $\mathbb{R}^2 \to \mathbb{R}$:



- Such a subdivision probably exists.
- But can we compute whether we are finished subdividing without doing a full CAD projection? (maybe not)

Multivariate Subdivision Algorithm

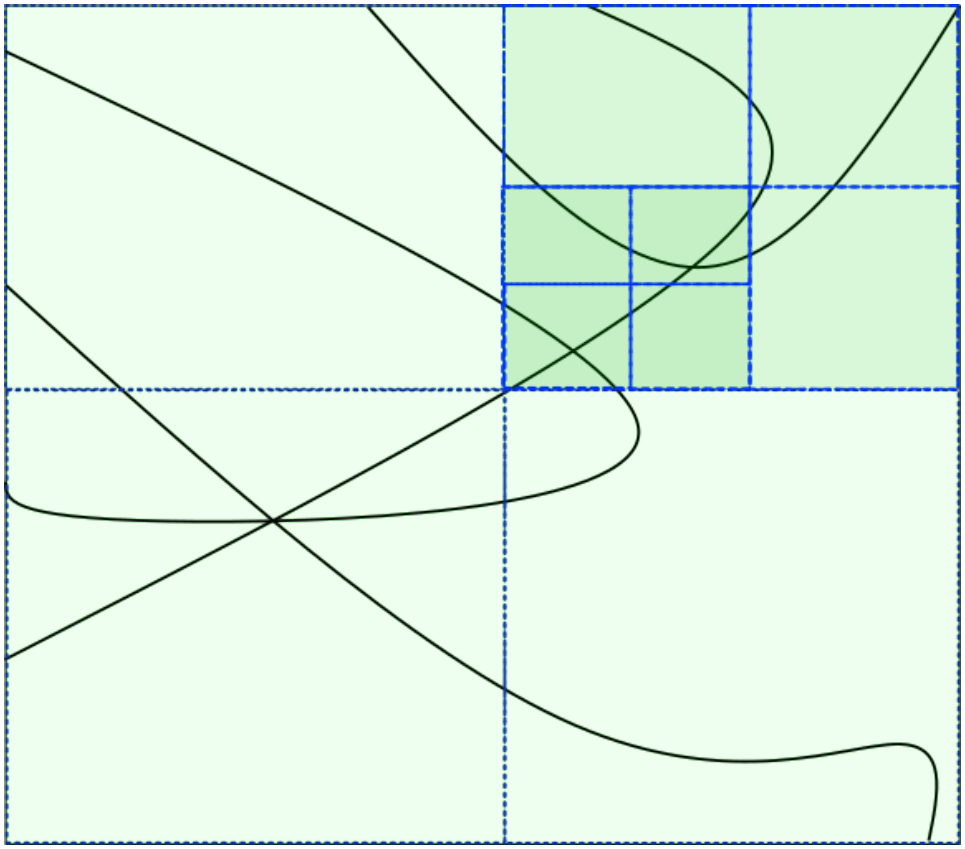
2-dimensional example: plot the zero sets of polynomials $\mathbb{R}^2 \to \mathbb{R}$:



- Such a subdivision probably exists.
- But can we compute whether we are finished subdividing without doing a full CAD projection? (maybe not)

Multivariate Subdivision Algorithm

2-dimensional example: plot the zero sets of polynomials $\mathbb{R}^2 \to \mathbb{R}$:



- Such a subdivision probably exists.
- But can we compute whether we are finished subdividing without doing a full CAD projection? (maybe not)

Nonlinear Arithmetic

We have implemented/formalized/proved several other algorithms for determining satisfiability of a system

$$p_1(x_1, \ldots, x_n) > 0 \wedge \cdots \wedge p_m(x_1, \ldots, x_m) \leq 0$$

This includes some approximation methods:

- Interval Arithmetic
- Bernstein Polynomials
- Affine Arithmetic

Bernstein Polynomials

A single-variable polynomial can always be written as a Bernstein polynomial:

$$p^*(x) = \sum_{i=0}^{n} a_i \binom{n}{i} x^i (1-x)^{n-i}$$

Useful Properties:

- $\min_i a_i \leq p^*(x) \leq \max_i a_i$ for all $x \in [0,1]$.
- $a_i \geq M$ for all i implies $p^*(x) \geq M$ for all $x \in [0,1]$.
- $p^*(0) = a_o$ and $p^*(1) = a_n$.

Thus, Bernstein polynomials can be used to prove and find counterexamples to polynomial systems of the form

$$p_1(x_1, \ldots, x_n) > 0 \wedge \cdots \wedge p_m(x_1, \ldots, x_m) \leq 0$$

Similar for interval arithmetic (the naive way to do this).

Numerical Methods

Numerical Approximation Methods

- Interval Arithmetic
- Bernstein Polynomials

Both of these methods give a *crude* estimate of a range of $p_i(x_1, \ldots, x_n)$. These estimates get *better* for small boxes

Solution: Keep subdividing a big box into smaller boxes until you can prove the result.

Subdivision

$$p(x) \geq 0 \quad \text{on} \quad [A, B]$$

$$\downarrow$$

$$p^*(x) \geq 0 \quad \text{on} \quad [0, 1]$$

$$p^*(x) \geq 0 \quad \text{on} \quad [0, \tfrac{1}{2}] \qquad\qquad p^*(x) \geq 0 \quad \text{on} \quad [\tfrac{1}{2}, 1]$$

left subdiv

right subdiv

$$p_L^*(x) \geq 0 \quad \text{on} \quad [0, 1] \qquad\qquad p_R^*(x) \geq 0 \quad \text{on} \quad [0, 1]$$

A Generic Branch and Bound Algorithm

Pseudo-Code:

```
branch_and_bound(simplify, evaluate, branch, subdivide, denorm, combine, prune,
      lex, gex, select, accumulate, maxd, obj, dom, acc, dirvars) : Output ≡
  let
    thisobj = simplify(obj),
    thisans = evaluate(dom, thisobj),
    thisacc = if none?(acc) then thisans
              else accumulate(val(acc), thisans) endif,
    thisexit = gex(dirvars, thisacc, thisans),
    thisout = mk_out(thisans, thisexit, length(dirvars), 0)
  in
    if length(dirvars) = maxd or lex(thisans) or thisexit or
       prune(dirvars, thisacc, thisans) then thisout
    else let
        (dir, v) = select(dirvars, thisacc, dom, thisobj),
        (obj_l, obj_r) = branch(v, thisobj),
        (obj_1, obj_2) = if dir then (obj_l, obj_r) else (obj_r, obj_l) endif,
        (dom_l, dom_r) = subdivide(v, dom),
        (dom_1, dom_2) = if dir then (dom_l, dom_r) else (dom_r, dom_l) endif,
        out_1 = branch_and_bound(simplify, evaluate, branch, subdivide, denorm,
                    combine, prune, lex, gex, select, accumulate, maxd,
                    obj_1, dom_1, thisacc, push((dir, v), dirvars))
      in
        if out_1.exit then
          mk_out(combine(v, denorm((dir, v), out_1.ans), thisans), true,
                 out_1.depth, out_1.splits + 1)
        else let
            newacc = accumulate(thisacc, out_1.ans),
            out_2 = branch_and_bound(simplify, evaluate, branch, subdivide, denorm,
                        combine, lex, gex, select, accumulate, maxd,
                        obj_2, dom_2, newacc, push((¬dir, v), dirvars))
            (out_l, out_r) = if dir then (out_1, out_2) else (out_2, out_1) endif,
            ans = combine(v, denorm((true, v), out_l.ans), denorm((false, v), out_r.ans))
          in
            mk_out(ans, out_2.exit, max(out_1.depth, out_2.depth),
                   out_1.splits + out_2.splits + 1)
        endif
    endif
```

This algorithm is generic. It works for any bounding method like interval arithmetic or Bernstein polynomials.

Formally proved correct in PVS.

A Generic Branch and Bound Algorithm

Using the branch and bound algorithm in PVS is *automated*

```
Heart(x1,x2,x3,x4,x5,x6,x7,x8): MACRO real =
  -x1*x6^3+3*x1*x6*x7^2-x3*x7^3+3*x3*x7*x6^2-x2*x5^3+3*x2*x5*x8^2-x4*x8^3+3*x4*x8*x5^2-0.9563453

Heart_forall_: LEMMA
  -0.1 <= x1 AND x1 <= 0.4 AND
  0.4 <= x2  AND x2 <= 1 AND
  -0.7 <= x3 AND x3 <= -0.4 AND
  -0.7 <= x4 AND x4 <= 0.4 AND
  0.1 <= x5  AND x5 <= 0.2 AND
  -0.1 <= x6 AND x6 <= 0.2 AND
  -0.3 <= x7 AND x7 <= 1.1 AND
  -1.1 <= x8 AND x8 <= -0.3 IMPLIES
  Heart(x1,x2,x3,x4,x5,x6,x7,x8) >= -1.7435

Heart_exists_: LEMMA
  EXISTS (x1,x2,x3,x4,x5,x6,x7,x8:real):
    -0.1 <= x1 AND x1 <= 0.4 AND
    0.4 <= x2  AND x2 <= 1 AND
    -0.7 <= x3 AND x3 <= -0.4 AND
    -0.7 <= x4 AND x4 <= 0.4 AND
    0.1 <= x5  AND x5 <= 0.2 AND
    -0.1 <= x6 AND x6 <= 0.2 AND
    -0.3 <= x7 AND x7 <= 1.1 AND
    -1.1 <= x8 AND x8 <= -0.3 AND
    Heart(x1,x2,x3,x4,x5,x6,x7,x8) <= -1.7434
```

These theorems can both be proved by simply typing "(bernstein)" or "(interval)" in PVS

Conclusion

- We depend heavily on built-in decision procedures for real arithmetic
- We have built our own in PVS
  - Bernstein polynomials
  - Interval arithmetic
  - Sturm and Tarski theorems
  - Sturm's theorem and subdivision
  - Affine arithmetic
  - Exact real arithmetic
- We would like a fast, verified multivariate CAD in PVS (Integrated in a sound way).
- Matt Damon used the PVS libraries

Questions

Questions?