Formally Verifying an Algorithm Based on Interval Arithmetic for Checking Transversality

Marcio Gameiro and Panagiotis Manolios

gameiro@math.gatech.edu, manolios@cc.gatech.edu

Georgia Institute of Technology
Atlanta, Georgia, 30332, USA
Outline

- Motivation (Analyze Differential Equations).
- Differential Equations.
- Polygon Based Algorithm.
- Transversality Checking.
- Interval Arithmetic.
- ACL2 Implementation.
- Conclusions.
- Future Work.
Motivation

- Differential equations widely used in sciences.
- No general analytical techniques are available.
  - Resort to numerics (imprecisions may arise).
- Want “rigorous” numerics (proofs).
  - Algorithms need to be correct.
  - Implementation needs to be correct.
- ACL2 well-suited for program verification.
- This work started as a Formal Methods class project.
Example:
\[
\begin{align*}
\dot{x} &= y \\
\dot{y} &= -x + \mu(1 - x^2)y
\end{align*}
\]

- **Analysis:** Is there a periodic orbit?, . . . .
- **Standard approach:** numerics on grids.
  - Numerical errors make this unsound.
  - Want soundness: every claim is true.
Polygon Based Algorithm

- An approach that uses numerics but is sound.

Triangulation

Directed graph

- Need transversality.
Strongly connected components

Transversality ⇒ proofs (via Conley Index).
Conley Index provides dynamics.

**Strengths:**
- Results are guaranteed to be correct.
- Use numerics, get proofs.
- Get qualitative dynamics info, not single orbits.

**Limitations:**
- Not guaranteed to capture all interesting dynamics.
- Only captures dynamics in a given region.
Mathematical Definitions

- $\dot{x} = f(x), \ x \in \mathbb{R}^n$ (Differential equation)
- $\varphi : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^n$ (Flow)
- $S$ is Invariant iff $S = \varphi(\mathbb{R}, S)$
- $Inv(N, \varphi) := \{x \in N \mid \varphi(\mathbb{R}, x) \subseteq N\}$ (Maximal invariant set)
- $N$ is an Isolating neighborhood iff $Inv(N, \varphi) \subseteq Int(N)$ and $N$ compact
- $N$ is Isolating block iff $\langle \forall x \in \partial N, t > 0 :: \varphi((-t, t), x) \not\subseteq N \rangle$
Mathematical Definitions 2

- For \( x \in \partial N \), \( \varphi((-t,t), x) \not\subseteq N \) is equivalent to:
  \[
  f(x) \cdot n(x) \neq 0 \text{ (transversality condition)}, \text{ where } n(x) \text{ is the normal vector to } \partial N \text{ at } x.
  \]

- \( N^- := \{ x \in N \mid \forall t > 0, \varphi((0, t), x) \not\subseteq N \} \) (Immediate Exit Set).

- If \( N \) is an isolating block, then \( CH_\ast(N) := H_\ast(N, N^-) \) (Conley Index).

- \( H_\ast(N, N^-) \) denotes the Relative homology groups.
Transversality

Need

\[ n \cdot f(u + \lambda(v - u)) \neq 0 \]
\[ \forall \lambda \in [0, 1]. \]

- Numerics need to be rigorous.
  - Only place rigorous numerics required.
- Need to check infinitely many points.
- Interval arithmetic solves both problems.
Interval Arithmetic

- We define interval arithmetic as follows:

\[ [x_1, x_2] + [y_1, y_2] := \{ x + y \mid x \in [x_1, x_2], y \in [y_1, y_2] \} \]

- Equivalently,

\[ [x_1, x_2] + [y_1, y_2] = [x_1 + y_1, x_2 + y_2] \]

- Implementation:

\[ [x_1, x_2] + [y_1, y_2] \subseteq [x_1, x_2] \oplus [y_1, y_2] \]

- Similarly for \(-\), \(\times\), \(\div\)
(defun intervalp (x1 x2)
   (and (real/rationalp x1)
        (real/rationalp x2)
        (<= x1 x2)))

(defun in (x x1 x2)
   (and (real/rationalp x)
        (intervalp x1 x2)
        (<= x1 x)
        (<= x x2)))

(encrypte
   (((i+ * * *) => (mv * *)))
   
   (local (defun i+ (x1 x2 y1 y2)

   ... ))

(defthm i+_ok
   (implies (and (in x x1 x2)
                 (in y y1 y2))
            (mv-let (z1 z2)
               (i+ x1 x2 y1 y2)
               (in (+ x y) z1 z2)))))
Transversality Check

\[ g(\lambda) := n \cdot f(u + \lambda(v - u)) \]

- Need to show \( \forall \lambda \in [0, 1] :: g(\lambda) \neq 0 \).
- Try to show \( 0 \not\in g([0, 1]) \)?
  - \( g([0, 1]) \) may be too large.
- Partition \([0, 1]\) and show \( 0 \not\in g([\lambda_i, \lambda_{i+1}]) \) for each subinterval.
(encapsulate
  (((vec_fld * *) => (mv * *))
   (((i_vec_fld * * * *) => (mv * * * *)))))

(local (defun vec_fld (x y)
   ... ))

(local (defun i_vec_fld (x1 x2 y1 y2)
   ... ))

(defthm vec_fld_ok
  (implies (and (in x x1 x2) (in y y1 y2))
    (mv-let (u1 u2 w1 w2) (i_vec_fld x1 x2 y1 y2)
      (mv-let (u w) (vec_fld x y)
        (and (in u u1 u2) (in w w1 w2))))))
ACL2 Implementation 2

(defun dot (x y u v)
  (+ (* x u) (* y v)))

(defun i_dot (x1 x2 y1 y2 u1 u2 v1 v2)
  (mv-let (p11 p12) (* x1 x2 u1 u2)
          (mv-let (p21 p22) (* y1 y2 v1 v2)
                   (mv-let (d1 d2) (+ p11 p12 p21 p22)
                            (mv d1 d2))))))

(defthm i-dot-ok
  (let ((idot (i-dot x1 x2 y1 y2 u1 u2 v1 v2)))
    (implies (and (in x x1 x2) (in y y1 y2)
                   (in u u1 u2) (in v v1 v2))
             (in (dot x y u v) (nth 0 idot) (nth 1 idot))))
(defun perp (x y)  
  (mv (* -1 y) x))

(defun i_perp (x1 x2 y1 y2)  
  (mv-let (ny1 ny2) (i* -1 -1 y1 y2)  
    (mv ny1 ny2 x1 x2)))

(defun normal_vec (u1 u2 v1 v2)  
  (mv-let (w1 w2) (perp (- v1 u1) (- v2 u2))  
    (mv w1 w2)))

(defun i_normal_vec (u1 u2 v1 v2)  
  (mv-let (x1 x2) (i- v1 v1 u1 u1)  
    (mv-let (y1 y2) (i- v2 v2 u2 u2)  
      (mv-let (w11 w12 w21 w22) (i_perp x1 x2 y1 y2)  
        (mv w11 w12 w21 w22)))))

Formally Verifying an Algorithm Based on Interval Arithmetic for Checking Transversality – p.16/20
(defun check-trans-lbda (u1 u2 v1 v2 lbda)
  (and (real/rationalp u1)
       (real/rationalp u2)
       (real/rationalp v1)
       (real/rationalp v2)
       (in lbda 0 1)
       (mv-let (n1 n2)
               (normal-vec u1 u2 v1 v2)
               (mv-let (y1 y2)
                       (edge-lbda u1 u2 v1 v2 lbda)
                       (mv-let (f1 f2)
                                (vec-fld y1 y2)
                                (not (equal (dot f1 f2 n1 n2)
                                            0))))))

(defun i-check-trans-lbda (u1 u2 v1 v2 l1 l2)
  ... )
(defthm edge_trans_f
  (implies (and (in lbda 0 1)
                 (unit-partition 1)
                 (real/rationalp-hyps u1 u2 v1 v2)
                 (i_check_trans u1 u2 v1 v2 1))
            (check_trans_lbda u1 u2 v1 v2 lbda)))

(defun real/rationalp-hyps (u1 u2 v1 v2)
  (and (real/rationalp u1) (real/rationalp u2)
       (real/rationalp v1) (real/rationalp v2)))

(defun i_check_trans (u1 u2 v1 v2 1)
  (if (endp (cddr 1))
      (i_check_trans_lbda u1 u2 v1 v2 (car 1) (cadr 1))
      (and (i_check_trans_lbda u1 u2 v1 v2 (car 1) (cadr 1))
           (i_check_trans u1 u2 v1 v2 (cdr 1))))
Conclusions

- Use ACL2 to analyze differential equations.
- Algorithm produces proofs.
- Use numerics to check transversality, but proofs needed: interval arithmetic.
- Formalized interval arithmetic in ACL2.
- Verified transversality algorithm in ACL2.
Future Work

- Incorporate interval arithmetic in ACL2 and ACL2(r).
  - Produce proofs via computation.
  - Allow computing with reals.
- Generalize to higher dimensions.
  - We only treated the 2 dimensional case.
- Implement and verify the program in ACL2.
  - We can then run the verified code.
  - Trust this much more than C implementation.