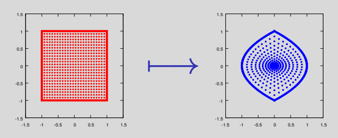
Wrapping in Exact Real Arithmetic *

$$\begin{tikzcd}
\mathbb{R} \arrow[r, dashed, "f_1"] & \mathbb{R} \arrow[r, dashed, "f_2"] & \mathbb{R} \\
\mathbb{I}^{\mathbb{N}} \arrow[u, dashed, "\varrho"'] \arrow[r, dashed, "\Gamma_1"'] & \mathbb{I}^{\mathbb{N}} \arrow[u, dashed, "\varrho"'] \arrow[r, dashed, "\Gamma_2"'] & \mathbb{I}^{\mathbb{N}} \arrow[u, dashed, "\varrho"']
\end{tikzcd}$$



Norbert Müller

Universität Trier

CIRM Luminy, 2016-01-12

*joint work with M. Korovina, IIS Novosibirsk and F. Brauße, Universität Trier

1 Computability on real numbers

2 Exact real arithmetic

3 Wrapping Sets

4 Taylor Models

5 Examples

6 Closing remarks

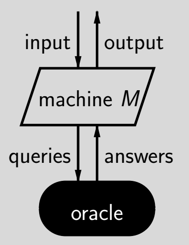
A real number $\boldsymbol{x}$ is usually represented as follows:

- use open intervals with dyadic endpoints

$$\mathbb{I} := \left\{ \left(\frac{m_1}{2^k}, \frac{m_2}{2^k} \right) \mid m_1, m_2 \in \mathbb{Z}, k \in \mathbb{N} \right\}$$

- aiming at oracle Turing machines
interpret word functions as sequences

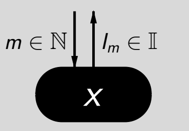
$$[\Sigma^* \to \Sigma^*] \quad \sim \quad [\mathbb{N} \to \mathbb{I}] \;=\; \mathbb{I}^{\mathbb{N}}$$



- define representation $\varrho : \subseteq \mathbb{I}^{\mathbb{N}} \to \mathbb{R}$:

$x \in \mathbb{R}$ is represented by $(I_m)_{m \in \mathbb{N}}$ iff

$$\lim_{m \to \infty} \operatorname{diam}(I_m) = 0 \quad \wedge \quad \bigcap_{m \in \mathbb{N}} I_m = \{x\}$$



A real function $\boldsymbol{f}$ is computed using a machine $\boldsymbol{M}$ as follows:

- If

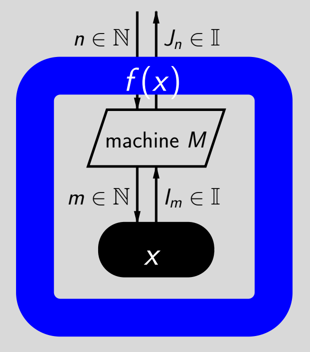
  $$\varrho\big((\boldsymbol{I_m})_{\boldsymbol{m}\in\mathbb{N}}\big) = \boldsymbol{x}$$

  and

  $$\boldsymbol{\Gamma_M} : (\boldsymbol{I_m})_{\boldsymbol{m}\in\mathbb{N}} \rightsquigarrow (\boldsymbol{J_n})_{\boldsymbol{n}\in\mathbb{N}}$$

  then

  $$\varrho\big((\boldsymbol{J_n})_{\boldsymbol{n}\in\mathbb{N}}\big) = \boldsymbol{f(x)}$$



Computable analysis (via ‘representations’):

$$\begin{tikzcd}
\mathbb{R} \arrow[r, dashed, "f_1"] & \mathbb{R} \arrow[r, dashed, "f_2"] & \mathbb{R} \\
\mathbb{I}^{\mathbb{N}} \arrow[u, dashed, "\varrho"'] \arrow[r, dashed, "\Gamma_1"'] & \mathbb{I}^{\mathbb{N}} \arrow[u, dashed, "\varrho"'] \arrow[r, dashed, "\Gamma_2"'] & \mathbb{I}^{\mathbb{N}} \arrow[u, dashed, "\varrho"']
\end{tikzcd}$$

Remember: Computable functions are **continuous!**

1. Computability on real numbers
2. Exact real arithmetic
3. Wrapping Sets
4. Taylor Models
5. Examples
6. Closing remarks

Wanted:
Implementation of real numbers on ‘real’ computers

- real numbers as abstract datatype
- real numbers as (atomic) objects

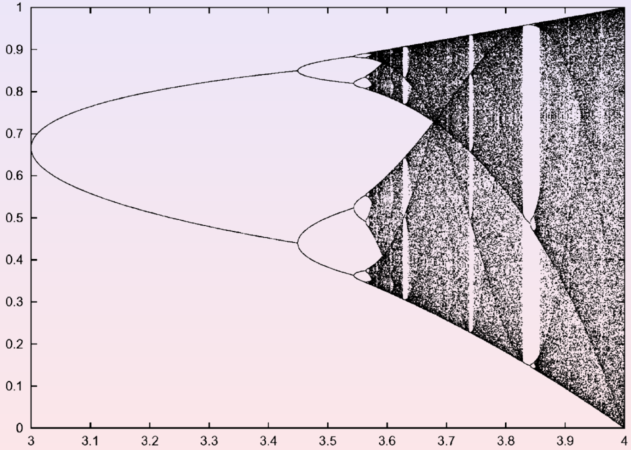
close at hand:

- $x \in \mathbb{R} \longleftrightarrow \lambda n.I_n \in \mathbb{I}^{\mathbb{N}}$
- so just implement $\lambda n.I_n$ in your favorite language
- with assertion

$$\lim_{n\to\infty} \operatorname{diam}(I_n) = 0 \quad \wedge \quad \{x\} = \bigcap_{n\in\mathbb{N}} I_n$$

Example: Logistic map

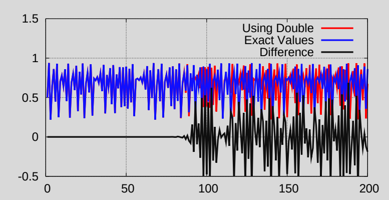
$$x_{n+1} = c \cdot x_n \cdot (1 - x_n) \qquad \text{for } x_0 \in (0,1), c \in (3,4)$$



Example: Logistic map

$$x_{k+1} = c \cdot x_k \cdot (1 - x_k) \qquad \text{for } x_0 = 0.5, c = 3.75$$

```
void itsyst(int k){

 REAL     x = 0.5;
 REAL     c = 3.75;

 for ( int i=1; i<=k; i++ ) {
  x = c * x * (1-x);
 }

 cout << x ;
}
```



(implemented in **iRRAM** library)

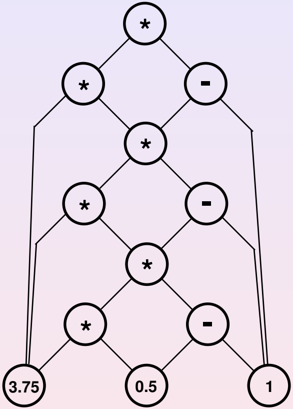
exact real arithmetic: ‘lazy’ approach using DAGs

```
REAL   x = 0.5;
REAL   c = 3.75;

for ( int i=1; i<=3; i++ )
   x= c*x*(1-x);

cout << x;
```

- easy(?) to extend: define new nodes...

- computation is lossless, no rounding applied
- quite memory intensive...

- ‘Lazy Evaluation’ (usually using MP-intervals)
- values are approximated, but only on demand
- evaluation bottom-up or top-down



(data structures behind variables represent exact values)

“Relax,
but don’t be too lazy.”

J. van der Hoeven, 2002

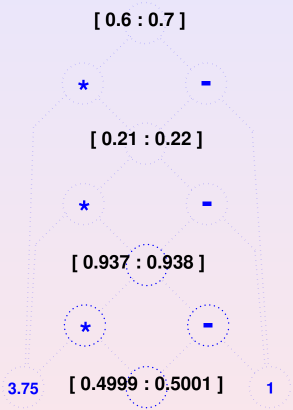
“Better be sloppy and fast
than precise and slow.”

A.B., MAP 2016

(... as long are you don’t make a mistake...)

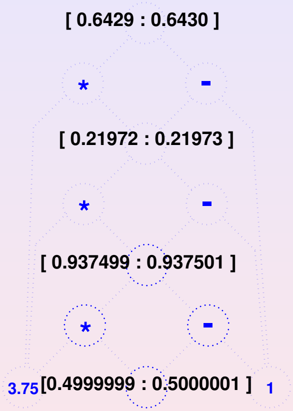
exact real arithmetic: ‘eager’ approach

```
REAL   x = 0.5;
REAL   c = 3.75;

for ( int i=1; i<=3; i++ )
   x= c*x*(1-x);

cout << x;
```



exact real arithmetic: ‘eager’ approach

```
REAL   x = 0.5;
REAL   c = 3.75;

for ( int i=1; i<=3; i++ )
   x= c*x*(1-x);

cout << x;
```



- computation is lossy, good for memory!
- iteration of computations!
- ‘exceptions’ are the rule...

- encapsulate I/O !
- multi-valuedness now is a problem...
- ‘decision history’, ‘multi-value-cache’

(data structures behind variables are only approximations)

Extension to further spaces:

easy question? trivial problem ?

- vectors $\mathbb{R}^k$ with maximum/euclidean norm $\rightsquigarrow$ **`vector<REAL>`**
- complex numbers $\mathbb{C}$ $\rightsquigarrow$ **`COMPLEX`**
- Sequences $\mathbb{R}^{\mathbb{N}}$ $\rightsquigarrow$ **`FUNCTION<REAL,int>`**
- Functions $f :\subseteq \mathbb{R} \to \mathbb{R}$ $\rightsquigarrow$ **`FUNCTION<REAL,REAL>`**
- Special function spaces [Thies15] like:
  - polynomials, $\rightsquigarrow$ **`POLY`**
  - analytical functions $\rightsquigarrow$ **`BA_ANA`**

How to implement? Lazy? Eager? Mixing both ways?

`iRRAM`: Only $\mathbb{R}^k$ and $\mathbb{C}$ are eager...

Higher order example: Taylor series

- consider $f : \mathbb{R} \to \mathbb{R}$ given by Taylor coefficients $(a_n)_{n\in\mathbb{N}}$:

$$f(x) = \sum_{n=0}^{\infty} a_n \cdot x^n$$

  - together with lower bound R for radius of convergence and
  - upper bound M for values $|f(x)|$ for $x \in \mathbb{C}$, $|x| \leq R$ or
  - upper bound M with $|a_n| \leq M \cdot R^{-n}$ (cf. Cauchy integral formula)

- then truncation error bounded by

$$|\sum_{k=n+1}^{\infty} a_k x^k| \leq \frac{M \cdot R}{R - |x|} \cdot \left(\frac{|x|}{R}\right)^{n+1}$$

- $\rightsquigarrow$ non-algebraic operator for *infinite* summation

Application e.g. for $f(x) = \sum_{n=0}^{\infty} 1 \cdot x^n \qquad (= \frac{1}{1-x})$

```
 REAL proc (const int& n){return REAL(1);}

...

 REAL R,M;

 FUNCTION <REAL,int> a;
 FUNCTION <REAL,REAL> f;

 a = proc; R = 1; M = 1;

 f = taylor_sum(a,R,M);

 cout << f(0.25);
```

1 Computability on real numbers

2 Exact real arithmetic

3 **Wrapping Sets**

4 Taylor Models

5 Examples

6 Closing remarks

Modified view on computations:

- Use vectors $\overline{x}_i$ from $\mathbb{R}^d$ as states during a ‘real’ computation.
- Each step is a function $f_i : \subseteq\mathbb{R}^d \to \mathbb{R}^d$ computed by some Γ_i.
- Use sequences $\overline{O}_{i.j}$ of approximating vectors, i.e.
  $\overline{x}_i = \varrho^d\left((O_{i.j})_{j\in\mathbb{N}}\right)$

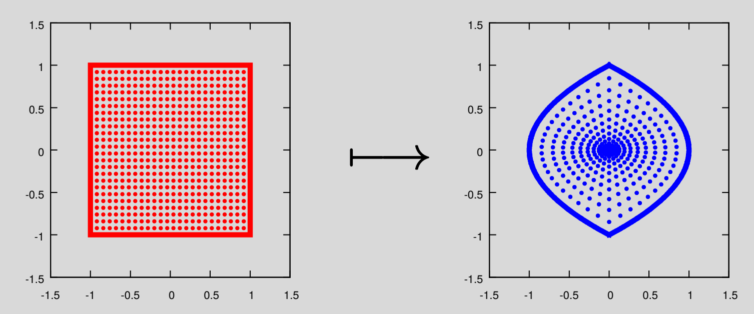
$$\begin{tikzcd}
\overline{x}_0 \arrow[r, mapsto, "f_0"] & \overline{x}_1 \arrow[r, mapsto, "f_1"] & \overline{x}_2 \arrow[r, mapsto, "f_2"] & \ldots \arrow[r, mapsto, "f_{n-1}"] & \overline{x}_n \\
\left.\begin{matrix}\vdots\\ O_{0,2}\\ O_{0,1}\\ O_{0,0}\end{matrix}\right\} \arrow[u] \arrow[r, mapsto, "\Gamma_0"] & \left\{\begin{matrix}\vdots\\ O_{1,2}\\ O_{1,1}\\ O_{1,0}\end{matrix}\right\} \arrow[u] \arrow[r, mapsto, "\Gamma_1"] & \left\{\begin{matrix}\vdots\\ O_{2,2}\\ O_{2,1}\\ O_{2,0}\end{matrix}\right\} \arrow[u] \arrow[r, mapsto, "\Gamma_2"] & \ldots \arrow[r, mapsto, "\Gamma_{n-1}"] & \left\{\begin{matrix}\vdots\\ O_{n,2}\\ O_{n,1}\\ O_{n,0}\end{matrix}\right. \arrow[u]
\end{tikzcd}$$

- Optimized for memory (like in **`iRRAM`**): Work line by line...

$$\begin{tikzcd}
O_{0,j} \arrow[r, mapsto, "\Gamma_0(j)"] & O_{1,j} \arrow[r, mapsto, "\Gamma_1(j)"] & O_{2,j} \arrow[r, mapsto, "\Gamma_2(j)"] & \ldots \arrow[r, mapsto, "\Gamma_{n-1}(j)"] & O_{n,j}
\end{tikzcd}$$

Necessary condition: $\boldsymbol{f_i(O_{i,j}) \subseteq O_{i+1,j}}$ ...
... but there is overestimation in interval vector computations:

- images of boxes $\boldsymbol{O} \in \mathbb{I}^{\boldsymbol{d}}$ aren't boxes again...
- example: function $\boldsymbol{f} : \mathbb{R}^2 \to \mathbb{R}^2$ with $\boldsymbol{f(x,y) = (x^2 - y^2, x \cdot y)}$



- 'wrapping' effects accumulate... so try to avoid or reduce them

Generalization:

- use ‘big’ topological space[†] M instead of $\mathbb{R}^d$
- e.g.

$$M = \mathbb{R} \cup \mathbb{R}^2 \cup \mathbb{R}^3 \cup \ldots \\ \ldots \cup \mathbb{C} \cup C(\mathbb{R}) \cup \ldots$$

Usual approach [Weihrauch,...]

- effective topological spaces (M, σ, ν)
- represent M via sequences $\sigma^{\mathbb{N}}$ for subbase σ
- ν enumerates σ
  - preimages of open sets are open: good for backward analysis
  - forward computation: apply continuous functions to open sets ...
    $\leadsto$ overestimation, wrapping effects...
- using product representations (like ϱ^d)
  for product spaces (like $\mathbb{R}^d$)

[†] metric, Hausdorff, perhaps even T_0

Replace open boxes $\mathbb{I}^d$:

Definition 3.1

A countable family $\mathcal{A} = \{\mathbf{A}_n \mid n \in \mathbb{N}\}$ *of sets* $\mathbf{A}_n \subseteq M$ *is wrapping iff* $\forall \mathbf{x} \in M\ \forall \varepsilon \in \mathbb{R}_{>0}$

$$\exists \mathbf{A} \in \mathcal{A},\ \ \mathrm{diam}(\mathbf{A}) \le \varepsilon \quad \wedge \quad \mathbf{x} \in \mathrm{int}(\mathbf{A})$$

Corresponding representation $\tau_{\mathcal{A}} : \subseteq \mathcal{A}^{\mathbb{N}} \to M$ *:*

$$\tau_{\mathcal{A}}(p) := \mathbf{x} \textit{ iff the sequence } p \in \mathcal{A}^{\mathbb{N}} \textit{ satisfies}$$
$$\lim_{n \in \mathbb{N}} \mathrm{diam}(p_n) = 0 \quad \wedge \quad \bigcap_{n \in \mathbb{N}} p_n = \{\mathbf{x}\}$$

Examples for $M = \mathbb{R}^d$:

- $\mathbb{I}^d$, closed boxes (also with point intervals), unions of boxes...
- DAGs, symbolic representations, ...
- Taylor models

Lemma 3.2

If $\mathcal{A}$ and $\mathcal{B}$ are wrapping, then $\tau_{\mathcal{A}}$ and $\tau_{\mathcal{B}}$ are topologically equivalent.

Basic idea of topological reduction $\tau_{\mathcal{A}} \leq \tau_{\mathcal{B}}$:

- W.l.o.g. $\boldsymbol{M} \in \mathcal{B}$
- For all $\boldsymbol{A} \in \mathcal{A}$, $\boldsymbol{n} \in \mathbb{N}$ there is $\boldsymbol{B} \in \mathcal{B}$ with

$$\boldsymbol{A} \subseteq \boldsymbol{B} \quad \wedge$$
$$\mathbf{diam}(\boldsymbol{B}\,) \leq \mathbf{inf}\{\mathbf{diam}(\boldsymbol{B}') \mid \boldsymbol{A} \subseteq \boldsymbol{B}' \in \mathcal{B}\} + \mathbf{2}^{-\boldsymbol{n}}$$

- Use arbitrary such function $\boldsymbol{w}^{\mathcal{B}}_{\mathcal{A}} : (\boldsymbol{A}, \boldsymbol{n}) \mapsto \boldsymbol{B}$.
- Define $\boldsymbol{W} : \mathcal{A}^{\mathbb{N}} \to \mathcal{B}^{\mathbb{N}}$:

$$\boldsymbol{W}(\boldsymbol{p}) := \boldsymbol{q} \quad \text{where} \quad \boldsymbol{q_n} := \boldsymbol{w}^{\mathcal{B}}_{\mathcal{A}}(\boldsymbol{p_n}, \boldsymbol{n})$$

- $\boldsymbol{W}$ is continuous, transformation stays strictly local!
- $\boldsymbol{W}$ is realizer for $\boldsymbol{id_M}$.

Special case:

Lemma 3.3

$\mathcal{A}$, $\mathcal{B}$ wrapping families:
Suppose there is a $((\nu_{\mathcal{A}}, \nu_{\mathbb{N}}), \nu_{\mathcal{B}})$-computable multivalued function

$$w_{\mathcal{A}}^{\mathcal{B}} : \mathcal{A} \times \mathbb{N} \rightrightarrows \mathcal{B}$$

such that for all $A \in \mathcal{A}$, $n \in \mathbb{N}$,

$$A \subseteq w_{\mathcal{A}}^{\mathcal{B}}(A, n) \wedge$$

$$\mathrm{diam}(w_{\mathcal{A}}^{\mathcal{B}}(A, n)\ ) \leq 2^{-n} + \inf\{\mathrm{diam}(B') \mid A \subseteq B' \in \mathcal{B}\}.$$

Then $\tau_{\mathcal{A}}$ is computably reducible to $\tau_{\mathcal{B}}$.

Special case $\boldsymbol{M} = \mathbb{R}^{\boldsymbol{d}}$:

- We always(!) have topological equivalence to $\varrho^{\boldsymbol{d}}$.
- In interesting cases we have computational equivalence to $\varrho^{\boldsymbol{d}}$.

$\rightsquigarrow$ wrapping is not important for computability ... but for efficiency!

- Computations will usually change only single component of a state in one step.
- Instead of general $\boldsymbol{f} : \subseteq\mathbb{R}^{\boldsymbol{d}} \to \mathbb{R}^{\boldsymbol{d}}$ we aim at $\boldsymbol{f}$ with

$$\boldsymbol{f}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{g}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \\ \vdots \\ \boldsymbol{x}_n \end{pmatrix}$$

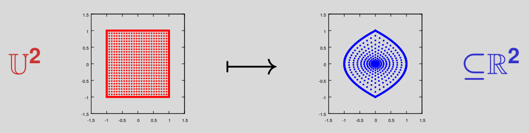
for a corresponding computable function $\boldsymbol{g} : \subseteq\mathbb{R}^{\boldsymbol{d}} \to \mathbb{R}$

1. Computability on real numbers
2. Exact real arithmetic
3. Wrapping Sets
4. **Taylor Models**
5. Examples
6. Closing remarks

Taylor models [Makino/Berz]:

- aim at $\mathbb{R}^{\boldsymbol{d}}$ with dynamical changeable $\boldsymbol{d}$
- use hypercube $\mathbb{U}^{\boldsymbol{k}}$ with $\mathbb{U} = [-1, 1]$, $\boldsymbol{k}$ independent from $\boldsymbol{d}$
- use vectors $\overline{\lambda} = (\lambda_1, \ldots, \lambda_k)$ of 'error symbols' $\lambda_i \in \mathbb{U}$
- consider $\mathbf{T}(\overline{\lambda}) = \sum_{\overline{n}} \overline{c}_{\overline{n}} \cdot \overline{\lambda}^{\overline{n}}$ of multivariate polynomials in $\overline{\lambda}$
- coefficients $\overline{c}_{\overline{n}}$ are vectors from $\mathbb{R}^{\boldsymbol{d}}$
- example with $\boldsymbol{d} = \boldsymbol{k} = \mathbf{2}$:

$$T(\overline{\lambda}) = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} \cdot \lambda_1 \cdot \lambda_2 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \cdot \lambda_1^2 + \begin{pmatrix} -1 \\ 0 \end{pmatrix} \cdot \lambda_2^2$$



$\rightsquigarrow$ Taylor models lead to wrapping families: $\mathbf{A} = \mathbf{T}(\mathbb{U}^{\boldsymbol{k}})$

Coefficient space $\mathbb{K}$ for components $\overline{c}_{\overline{n}} = \begin{pmatrix} c_{\overline{n},1} \\ \vdots \\ c_{\overline{n},d} \end{pmatrix}$?

- in practice: coefficients $\overline{c}_{\overline{n},i}$ are in double precision
- generalize to:
  - dyadic numbers $\mathbb{D}$
  - rational numbers $\mathbb{Q}$
  - computable real numbers $\mathbb{R}_c$
  - computable complex numbers $\mathbb{C}_c$
- further generalizations:
  - each/some $c_{\overline{n},i}$ might be an interval itself
- in practice: coefficient $\overline{c}_{\overline{0}}$ is usually vector of intervals

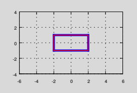
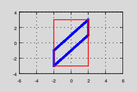
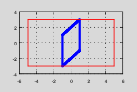
Variants of Taylor models:

- Affine arithmetic:
  order **1**, only $\boldsymbol{c_{\mathbf{0},i}}$ as non-point intervals
- Generalized interval arithmetic:
  order **1**, all $\boldsymbol{c_{\mathbf{n},i}}$ are arbitrary intervals
- Classical Taylor models:
  arbitrary order, only $\boldsymbol{c_{\mathbf{0},i}}$ as non-point intervals
- Interval Taylor models:
  arbitrary order, all $\boldsymbol{c_{\mathbf{n},i}}$ are arbitrary intervals

$\leadsto$ all versions of Taylor models yield same notion of computability!

Functions on $\mathbb{R}^d$ are implemented as transformations of polynomials:

- Addition/subtraction on $\mathbb{R}^d \rightsquigarrow$ polynomial addition/subtraction
- Example computation: `x=...; y=...; y=x+y; x=y-x;`
use $d = 2$, here with linear Taylor model and $k = 2$:

	interval	Taylor model	
$x = \ldots;$ $y = \ldots;$	$[-2,2]$ $[-1,1]$	$\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix}\cdot\lambda_1 + \begin{pmatrix}2\\0\end{pmatrix}\cdot\lambda_2$	
$y = x + y;$	$[-2,2]$ $[-3,3]$	$\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix}\cdot\lambda_1 + \begin{pmatrix}2\\2\end{pmatrix}\cdot\lambda_2$	
$x = y - x;$	$[-5,5]$ $[-3,3]$	$\begin{pmatrix}0\\0\end{pmatrix} + \begin{pmatrix}1\\1\end{pmatrix}\cdot\lambda_1 + \begin{pmatrix}0\\2\end{pmatrix}\cdot\lambda_2$	

- here there is no overestimation
- functional dependencies are completely retained...

- multiplication:
  $\leadsto$ polynomial multiplication (increasing degrees!)
- further functions f:
  $\leadsto$ substitute (truncated) Taylor series $f(x) = \sum_{n=0}^{\infty} a_n x^n$ series
  - given $x \sim \sum c_{\overline{k}} \cdot \overline{\lambda}^{\overline{k}}$
  - implement $y \sim f(x)$ as

$$\sum_{n=0}^{m} a_n \left(\sum c_{\overline{k}} \cdot \overline{\lambda}^{\overline{k}} \right)^n$$

  - determine error interval $[-\varepsilon, \varepsilon]$ due to truncation:

$$\left| \sum_{n=m+1}^{\infty} a_n \left(\sum c_{\overline{k}} \cdot \overline{\lambda}^{\overline{k}} \right)^n \right| \leq \varepsilon$$

$\leadsto$ allow interval coefficients at least for $\overline{c}_{\overline{0}}$
$\leadsto$ or each time add monomial $|\varepsilon| \cdot \lambda_{new}$ with new error symbol λ_{new}

Degrees grow very fast, ‘rounding’ to lower degree necessary!

Important functions for Taylor models:

- Sweeping:
  Reduce degrees of monomials by replacing error symbols with intervals

$$\boldsymbol{c} \cdot \lambda_1 \cdot \lambda_2 \quad \rightsquigarrow \quad \begin{cases} \boldsymbol{c} \cdot \mathbb{U} & \text{or} \\ \boldsymbol{c} \cdot \mathbb{U} \cdot \lambda_1 & \text{or} \\ \boldsymbol{c} \cdot \mathbb{U} \cdot \lambda_2 & \end{cases}$$

- Polishing:
  Sweep + introduce new error variables

$$\boldsymbol{c} = [\boldsymbol{d} \pm \varepsilon] \quad \rightsquigarrow \quad \boldsymbol{d} + \varepsilon \lambda_{\boldsymbol{new}}$$

$\rightsquigarrow$ Sweeping and polishing implement the identity function on $\mathbb{R}^d$

$\rightsquigarrow$ Sweeping and polishing reduce the internal data structure

Taylor models in **`iRRAM`** library:

- Generalized interval arithmetic, order **1**, interval coefficients
- Two real datatypes: **`REAL`**, **`TM`**
- Constructor **`REAL(TM T)`** sweeps **`T`** to basic coefficient used as interval for the **`REAL`** variable.
- Constructor **`TM(REAL R)`** takes interval from **`R`**, degree 0.
- Arithmetic on **`TM`** includes automatic sweeping for degree $\geq$**2**, reduces degree to linear.
- Polishing (combined with sweeping) must be triggered manually.

Current version:

- Prototype, only addition/subtraction/multiplication/polish
- $\sim$ **500** lines of code

1 Computability on real numbers

2 Exact real arithmetic

3 Wrapping Sets

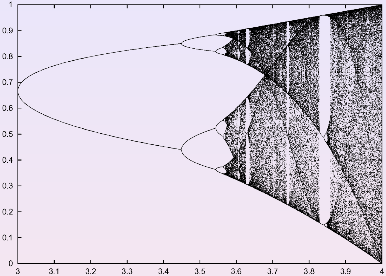
4 Taylor Models

5 Examples

6 Closing remarks

Example: Logistic map using Taylor models in iRRAM

```
void itsyst(REAL& c, int n){
  TM      x = REAL(0.125);
  for ( int i=0; i<=n; i++ ) {
    TM::polish(x);
    x = x * c * (REAL(1)-x);
  }
  cout << REAL(x) << "\n";
}
```



	Data type **TM**				Data type **REAL**			
c	n=10000		n=100000		n=10000		n=100000	
	time [s]	precision [bits]	time [s]	precision [bits]	time [s]	precision [bits]	time [s]	precision [bits]
3.125	0.09	double	0.90	double	1.08	18581	266	175466
3.56982421875	0.09	double	0.94	double	0.85	18581	363	219405
3.75	0.64	5894	115	57301	1.60	23299	400	219405
3.82	0.75	7440	148	71699	1.38	23299	340	219405
3.830078125	0.09	double	0.92	double	1.40	23299	337	219405
3.84	0.09	136	0.89	136	1.46	23299	354	219405

Example: Van der Pol oscillator, discretized

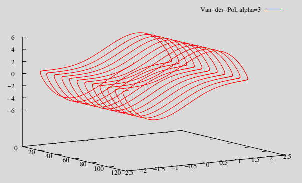
- nonlinear differential equation, $d = 2$

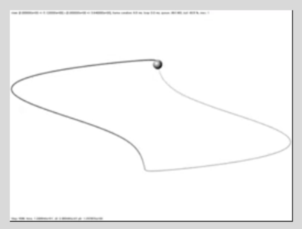
$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= \alpha y - x - \alpha x^2 y \end{aligned}$$

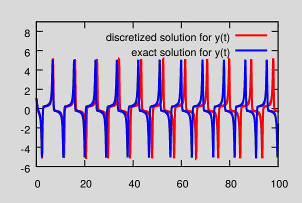
- using $\alpha = 3$
- initial value $w_0 = (1,1)$ at $t_0 = 0$

- discretized with $\Delta t = 0.01$ to

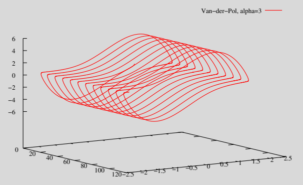
$$\begin{aligned} x_{n+1} &= x_n + \Delta t \cdot y_n \\ y_{n+1} &= y_n + \Delta t \cdot (\alpha y_n - x_n - \alpha x_n^2 y_n) \end{aligned}$$







Example: Van der Pol oscillator, discretized



```
std::vector<TM> x, x_new;
x_new.push_back(TM(REAL(1)));
x_new.push_back(TM(REAL(1)));

REAL alpha = 3; REAL t = 0.01;

for( int i = 0; i <= n ; i++){
   x = x_new;
   TM::polish(x);
   cout << setRwidth(15);
   cout << i*t <<" " << REAL(x[0]) <<" " << REAL(x[1]) <<"\n";

   x_new[0] = x[0] +  x[1]*t;
   x_new[1] = x[1] + (x[1]*alpha - x[0] - x[0]*x[0]*x[1]*alpha)*t;
}
```

		Data type TM		Data type REAL	
t_{end}	n	time [s]	precision [bits]	time [s]	precision [bits]
10	**1 000**	0.05	double	0.01	136
100	**10 000**	0.42	double	0.18	1737
1 000	**100 000**	4.6	136	6.9	14807
10 000	**1 000 000**	32	136	2395	175466
100 000	**10 000 000**	305	136	-	-

Example: Van der Pol oscillator, exact

Part I: Taylor series

- Consider sequence Taylor coefficients $(a_n)_{n\in\mathbb{N}}$ together with pair R, M for $|a_n| \leq M \cdot R^{-n}$

$\rightsquigarrow$ operator for *infinite* summation, transparent for Taylor models:

- Use Taylor model arithmetic to partial sums $S_{n,x}$ and error bounds

$$S_{n,x} := \sum_{k=0}^{n} a_k x^k \qquad E_{n,x} := \frac{M \cdot R}{R - |x|} \cdot \left(\frac{|x|}{R}\right)^{n+1}$$

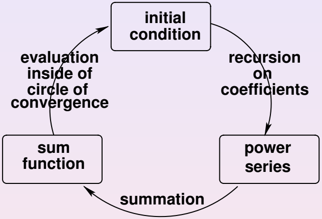
  until $E_{n,x}$ is 'small enough'

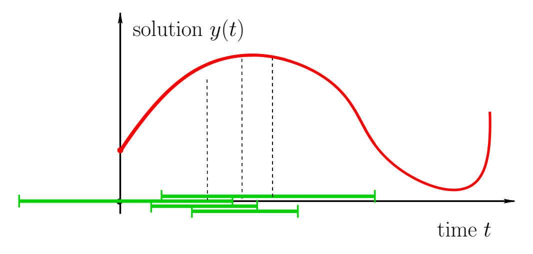
- then the eager approximation is $S_{n,x} + [0 \pm E_{n,x}]$

```
FUNCTION <TM,int> a = ...;
REAL R = ...; REAL M = ...; TM x = ...;
FUNCTION<TM,TM> f = taylor_sum(a,R,M);
cout << f(x);
```

Example: Van der Pol oscillator, exact

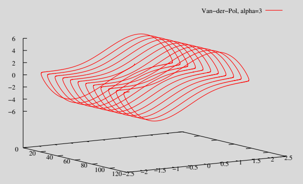
Part II: power series method, iterated:


initial condition
recursion on coefficients
power series
summation
sum function
evaluation inside of circle of convergence

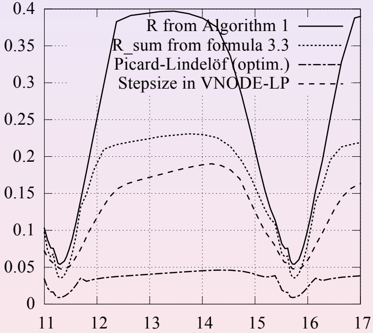


- radii of convergence are finite (unless system is linear)
- similar to analytic continuation, but finite(!) states $\boldsymbol{w_i}$ at $\boldsymbol{t_i}$
- again *polish* states $\boldsymbol{w_i}$ at times $\boldsymbol{t_i}$

Example: Van der Pol oscillator, exact



compute solution at t_{end} with **22** decimals:



t_{end}	Taylor models time [s]	 prec [bits]	intervals time [s]	 prec [bits]	 prec/t_{end} [bits]
8.25	56	242	33	242	29.3
15.75	108	242	124	375	23.8
23.75	153	242	272	541	22.8
34.00	232	242	1301	1008	29.6
63.00	412	242	2848	1332	21.1
83.50	572	242	5562	1737	20.8
109.25	761	242	10470	2242	20.5
140.75	924	242	21354	2876	20.4
200.00	1680	242			
250.00	2100	242			
300.00	2520	242			
350.00	2940	242			
500.00	3883	242			

`VNODE-LP`: 0.2s for $t_{\text{end}} = \mathbf{100}$, $\sim$ **12** decimals

1. Computability on real numbers
2. Exact real arithmetic
3. Wrapping Sets
4. Taylor Models
5. Examples
6. Closing remarks

Todo:

- enhancements
  - Taylor model versions of further functions
  - Taylor model versions of limit operators
- optimizations
  - improve sweeping/polishing
  - try higher order Taylor models
  - specific treatment of symbols of form λ^2
- precise complexity analysis
  - can we reliably predict the effects of Taylor models?

Thank you for your attention!

Questions? Remarks?