Grassmannians in Exact Real Computation

Seokbin Lee, Martin Ziegler

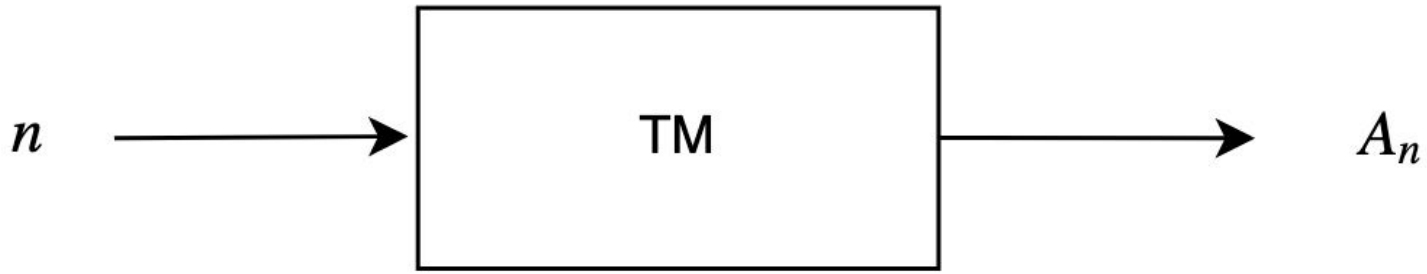
Korea Advanced Institute of Science and Technology

16th May 2019

Exact Real Computation (1/4)

Definition.

A number $x \in \mathbb{R}$ is computable if there is a computable sequence of rational numbers (A_n) such that $|x - A_n| < 2^{-n}$ for all $n \in \mathbb{N}$.



Exact Real Computation (2/4)

Exact Real Computation aims to emphasize the reals as encodings of sequences.

Exact Real Computation (2/4)

Exact Real Computation aims to emphasize the reals as encodings of sequences.

Definition.

In Exact Real Computation, the *representation* of $r \in \mathbb{R}$ is an infinite encoding of a rational sequence (A_n) where $|r - A_n| < 2^{-n}$.

Exact Real Computation (2/4)

Exact Real Computation aims to emphasize the reals as encodings of sequences.

Definition.

In Exact Real Computation, the *representation* of $r \in \mathbb{R}$ is an infinite encoding of a rational sequence (A_n) where $|r - A_n| < 2^{-n}$.

This construction has a caveat:

Fact.

Testing inequality "$x \neq y$" for two numbers x and y is equivalent to the Halting problem, so is undecidable. In particular, a test "$x > 0$" cannot return in the case $x = 0$.

Exact Real Computation (2/4)

Exact Real Computation aims to emphasize the reals as encodings of sequences.

Definition.

In Exact Real Computation, the *representation* of $r \in \mathbb{R}$ is an infinite encoding of a rational sequence (A_n) where $|r - A_n| < 2^{-n}$.

This construction has a caveat:

Fact.

Testing inequality "$x \neq y$" for two numbers x and y is equivalent to the Halting problem, so is undecidable. In particular, a test "$x > 0$" cannot return in the case $x = 0$.

However, this construction also has a merit...

Exact Real Computation (3/4)

Lemma (F. Braußе, P. Collins, J. Kanig, S. Kim, M. Konečný, G. Lee, N, Müller, E. Neumann, S. Park, N. Preining, M. Ziegler, 2016).

The language of computable numbers is Turing-complete; a function is computable iff it can be expressed in Exact Real Computation. Similarly, the language of functionals is also Turing complete.

Exact Real Computation (3/4)

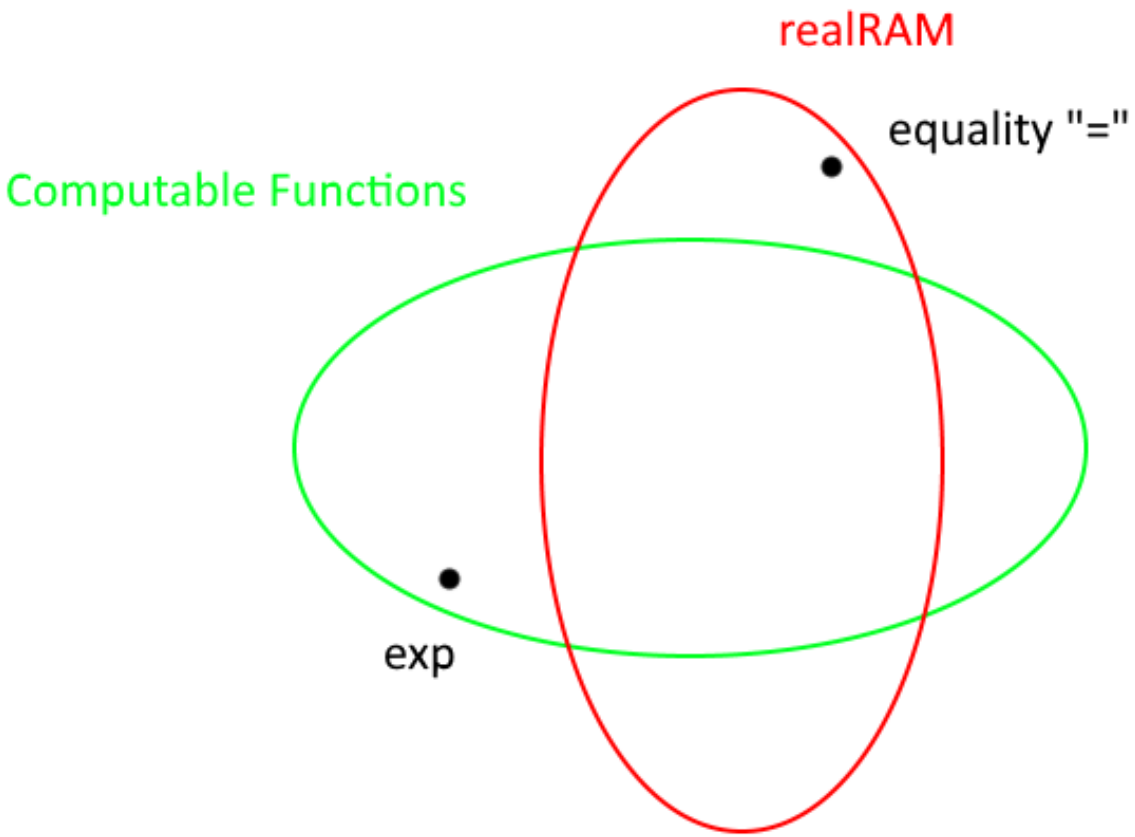
Lemma (F. Braußе, P. Collins, J. Kanig, S. Kim, M. Konečný, G. Lee, N, Müller, E. Neumann, S. Park, N. Preining, M. Ziegler, 2016).

The language of computable numbers is Turing-complete; a function is computable iff it can be expressed in Exact Real Computation. Similarly, the language of functionals is also Turing complete.

This differs from other paradigms such as realRAM (BSS machine), which does not have Turing-completeness.

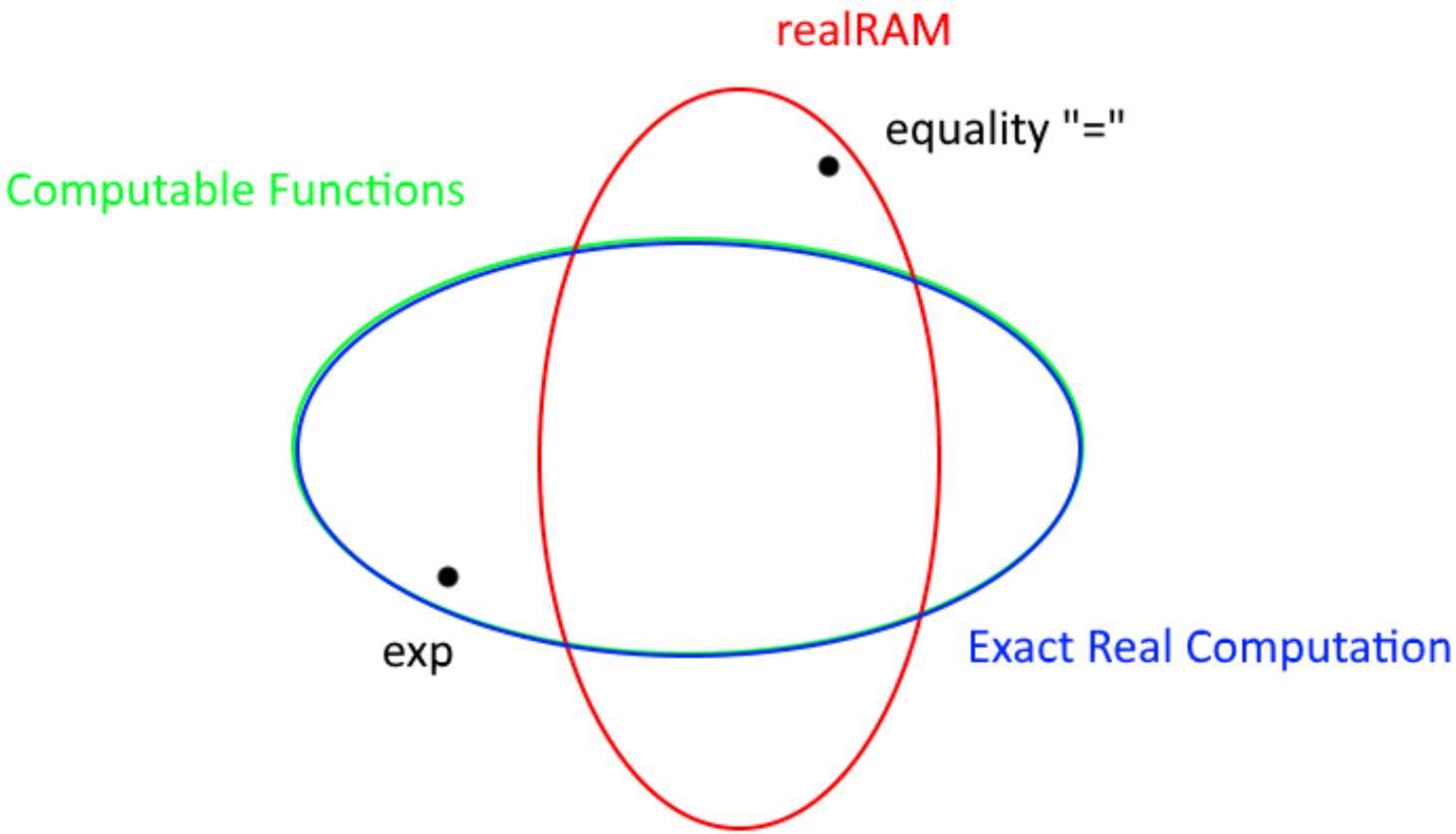
Exact Real Computation (4/4)

The following diagram illustrates this point.



Exact Real Computation (4/4)

The following diagram illustrates this point.



Grassmannians

Definition.

A *Grassmannian* $Gr(k, V)$ is the set of all k-dimensional linear subspaces of a vector space V.

Grassmannians

Definition.

A *Grassmannian* $Gr(k, V)$ is the set of all k-dimensional linear subspaces of a vector space V.

Example.

The Grassmannian $Gr(1, \mathbb{R}^3)$ is the set of all lines in $\mathbb{R}^3$ passing through the origin. The Grassmannian $Gr(n-1, \mathbb{R}^n)$ is the set of hyperplanes in $\mathbb{R}^n$, passing through the origin.

Grassmannians

Definition.

A *Grassmannian* $Gr(k, V)$ is the set of all k-dimensional linear subspaces of a vector space V.

Example.

The Grassmannian $Gr(1, \mathbb{R}^3)$ is the set of all lines in $\mathbb{R}^3$ passing through the origin. The Grassmannian $Gr(n-1, \mathbb{R}^n)$ is the set of hyperplanes in $\mathbb{R}^n$, passing through the origin.

Since we are interested mainly in $\mathbb{R}^n$ (and in particular in $\mathbb{R}^3$), we will denote $Gr(k, n)$ to mean $Gr(k, \mathbb{R}^n)$.

Representation of Grassmannian elements (1/2)

In the case of Grassmannians in $\mathbb{R}^3$, we need to consider only lines ($Gr(1,3)$) and planes ($Gr(2,3)$).

Representation of Grassmannian elements (1/2)

In the case of Grassmannians in $\mathbb{R}^3$, we need to consider only lines ($Gr(1,3)$) and planes ($Gr(2,3)$). These are projective spaces, so can be parametrized by a single vector.

Representation of Grassmannian elements (1/2)

In the case of Grassmannians in $\mathbb{R}^3$, we need to consider only lines ($Gr(1,3)$) and planes ($Gr(2,3)$). These are projective spaces, so can be parametrized by a single vector.
This is not true for higher dimensions - $Gr(2,4)$ is not projective space!

Representation of Grassmannian elements (2/2)

An intuitive way of representing Grassmannian elements is to consider a basis for the subspace, and encoding the basis as a set (in a suitable ADT).

Representation of Grassmannian elements (2/2)

An intuitive way of representing Grassmannian elements is to consider a basis for the subspace, and encoding the basis as a set (in a suitable ADT).
This generalizes better, but also has problems:

Representation of Grassmannian elements (2/2)

An intuitive way of representing Grassmannian elements is to consider a basis for the subspace, and encoding the basis as a set (in a suitable ADT).
This generalizes better, but also has problems:
- The basis needs to be linearly independent;

Representation of Grassmannian elements (2/2)

An intuitive way of representing Grassmannian elements is to consider a basis for the subspace, and encoding the basis as a set (in a suitable ADT).
This generalizes better, but also has problems:
- The basis needs to be linearly independent;
- Different bases may represent the same Grassmannian element;

Representation of Grassmannian elements (2/2)

An intuitive way of representing Grassmannian elements is to consider a basis for the subspace, and encoding the basis as a set (in a suitable ADT).
This generalizes better, but also has problems:
- The basis needs to be linearly independent;
- Different bases may represent the same Grassmannian element;
- The size of the basis can change, requiring dynamic memory allocation; as such, the size of the basis will also need to be stored.

Operations on Grassmannian elements

We are interested in operations on Grassmannian elements.

Operations on Grassmannian elements

We are interested in operations on Grassmannian elements.

Definition.

Given Grassmannian elements A and B, the set $A + B = \{a + b : a \in A, b \in B\}$ is the Minkowski sum of A and B.

Operations on Grassmannian elements

We are interested in operations on Grassmannian elements.

Definition.

Given Grassmannian elements A and B, the set $A+B = \{a+b : a \in A, b \in B\}$ is the Minkowski sum of A and B.

Definition.

Given Grassmannian elements A and B, the set $A \cap B$ is the intersection of A and B.

Operations on Grassmannian elements

We are interested in operations on Grassmannian elements.

Definition.

Given Grassmannian elements A and B, the set $A + B = \{a + b : a \in A, b \in B\}$ is the Minkowski sum of A and B.

Definition.

Given Grassmannian elements A and B, the set $A \cap B$ is the intersection of A and B.

Definition.

Given a Grassmannian element A, its orthogonal complement is the set $A^{\perp} = \{x : \forall a \in A, x \perp a\}$.

Specifications

The specification on operations involving two elements (addition, intersection) has a non-triviality:

Lemma.

Testing inequality in Exact Real Computation is equivalent to the Halting problem.

Corollary.

Testing inequality of two Grassmannian elements is undecidable (though semi-decidable).

Specifications

The specification on operations involving two elements (addition, intersection) has a non-triviality:

Lemma.

Testing inequality in Exact Real Computation is equivalent to the Halting problem.

Corollary.

Testing inequality of two Grassmannian elements is undecidable (though semi-decidable).

Because of this, we must *specify* that the supplied arguments are distinct, and *assert* within the algorithm that they are indeed unequal (though without a fundamental way of doing so).

Algorithms for operations in 3D (1/3)

We observe the following fact.

Fact.

Given a line in $Gr(1,3)$ and a plane in $Gr(2,3)$, testing whether the line is not contained in the plane is undecidable (though semi-decidable).

Algorithms for operations in 3D (1/3)

We observe the following fact.

Fact.

Given a line in $Gr(1,3)$ and a plane in $Gr(2,3)$, testing whether the line is not contained in the plane is undecidable (though semi-decidable).

Given the appropriate specifications, the algorithm for addition can be done via case-checking for the dimensions.

Algorithm 1 Minkowski Addition

1: **procedure** ADD(G_1, G_2)
2: **if** $dim(G_1) > dim(G_2)$ **then**
3: Swap G_1 and G_2
4: **if** $dim(G_2) = 3$ **then return** $\mathbb{R}^3$
5: **else if** $dim(G_1) = 0$ **then return** G_2
6: Assert $G_1 \not\subseteq G_2$
7: **if** $dim(G_1) = 2$ **then return** $\mathbb{R}^3$
8: **else if** $dim(G_2) = 1$ **then return** $G_1 \cup G_2$ (as bases)
9: **else return** $\mathbb{R}^3$ ▷ G_1 is a line, G_2 a plane

Algorithms for operations in 3D (2/3)

Similarly, we can devise an algorithm for the intersection of two Grassmannian elements.

Algorithm 3 Intersection

```
    procedure INTERSECT(G_1, G_2)
        if dim(G_1) > dim(G_2) then
3:          Swap G_1 and G_2
        if dim(G_2) = 3 then return G_1
        else if dim(G_1) = 0 then return ∅
6:      Assert G_1 ⊈ G_2
        if dim(G_1) = 2 then return G_1 ∩ G_2 (as bases)
        else if dim(G_2) = 1 then return ∅
9:      else return ∅                          ▷ G_1 is a line, G_2 a plane
```

For orthogonal complements, the outer product and Gram-Schmidt orthonormalization procedure handle most of the work...

Algorithms for operations in 3D (3/3)

So far, most of the work has been done by case-checking dimensions; does not scale easily with higher dimensions!

Algorithms for operations in 3D (3/3)

So far, most of the work has been done by case-checking dimensions; does not scale easily with higher dimensions!
An idea is to *specify* the supplement of another variable k that indicates the desired dimension of the result of the operations.

Algorithms for operations in 3D (3/3)

So far, most of the work has been done by case-checking dimensions; does not scale easily with higher dimensions!
An idea is to *specify* the supplement of another variable k that indicates the desired dimension of the result of the operations.
For example, consider lines $L_1, L_2 \in Gr(1,3)$.

Algorithms for operations in 3D (3/3)

So far, most of the work has been done by case-checking dimensions; does not scale easily with higher dimensions!
An idea is to *specify* the supplement of another variable k that indicates the desired dimension of the result of the operations.
For example, consider lines $L_1, L_2 \in Gr(1,3)$.
Calling ADD$(L_1, L_2, k = 2)$ indicates that the addition will yield a plane;
Calling ADD$(L_1, L_2, k = 2)$ indicates that the addition will yield a line.
This eliminates case-checking but requires general computations with bases!

Remaining Questions

- How do we specify the data type for bases of Grassmannian elements?

Remaining Questions

- How do we specify the data type for bases of Grassmannian elements?
- How to compute unions and intersections of subspaces with bases?

Remaining Questions

- How do we specify the data type for bases of Grassmannian elements?
- How to compute unions and intersections of subspaces with bases?
- What about the "projection" operation?

Definition.

Given a Grassmannian element S and a vector x, the projection of x onto S is the vector $y \in S$ such that $x - y$ is orthogonal to y.

Remaining Questions

- How do we specify the data type for bases of Grassmannian elements?
- How to compute unions and intersections of subspaces with bases?
- What about the "projection" operation?

Definition.

Given a Grassmannian element S and a vector x, the projection of x onto S is the vector $y \in S$ such that $x - y$ is orthogonal to y.

- Any challenges when implementing in iRRAM?

Thank you!