Mapping the space of Programs

Searching the space of Languages

Fritz Obermeyer

Department of Mathematics
Carnegie-Mellon University

2007:07:09
Outline

The space of programs
  An algebraic approach
  A computational approach
  A geometric approach

The space of languages

Applications

Summary
Pick a programming language.
Pick a programming language.

What are the simplest few programs?
Pick a simple programming langauge.

What are the simplest few programs?
Pick a simple programming language.

What are the simplest few program behaviors?
Pick a simple programming language family.

What are the simplest few program behaviors?
Pick a simple programming language family.

What are the simplest few program behaviors?

What are the simplest programming languages?
Pick a simple programming language family.

Combinators / $\lambda$-Calculus

What are the simplest few program behaviors?

What are the simplest programming languages?
Pick a simple programming language family.

Combinators / $\lambda$-Calculus

What are the simplest few program behaviors?

Map space of programs

What are the simplest programming languages?
Pick a simple programming language family.

Combinators / $\lambda$-Calculus

What are the simplest few program behaviors?

Map space of programs

What are the simplest programming languages?

Learn from examples
Programming as Algebra

What can we do with programs?

▶ apply one program to another
▶ compose programs
▶ copy programs
▶ permute arguments
▶ ignore arguments
▶ run two programs at once

This is combinatory algebra.
Programming as Algebra

What can we do with programs?

- apply one program to another

\[ f(x) \]

or just

\[ f \ x \]
Programming as Algebra

What can we do with programs?

- apply one program to another
- compose programs

\[ \lambda x. f(g\ x) \]
Programming as Algebra

What can we do with programs?

- apply one program to another
- compose programs
- copy programs

\( \lambda x.f \ x \ x \)
Programming as Algebra

What can we do with programs?

- apply one program to another
- compose programs
- copy programs
- permute arguments

\[ \lambda x, y. f \ y \ x \]
Programming as Algebra

What can we do with programs?

- apply one program to another
- compose programs
- copy programs
- permute arguments
- ignore arguments
Programming as Algebra

What can we do with programs?

- apply one program to another
- compose programs
- copy programs
- permute arguments
- ignore arguments
- run two programs at once

x | y

concurrency or non-determinism
Programming as Algebra

What can we do with programs?

▶ apply one program to another
▶ compose programs
▶ copy programs
▶ permute arguments
▶ ignore arguments
▶ run two programs at once

This is combinatory algebra.
Programming as Algebra

What can we do with programs?

- apply one program to another
- compose programs
- copy programs
- permute arguments
- ignore arguments
- run two programs at once

This is combinatory algebra.
Big Basis?
Big Basis?

not really:
Big Basis?

not really:

consider program

Behavior
Program Behavior: an abstract view

Behavior space is **denser** than program space.
Program Behavior: an abstract view

Behavior space is denser than program space. → we can build a bigger map.
Program Behavior: an abstract view

Behavior space is denser than program space. → we can build a bigger map.

When do two programs behave the same?
Program Behavior: an abstract view

Behavior space is denser than program space. → we can build a bigger map.

When do two programs behave the same?

- all programs that do nothing are equivalent.
Program Behavior: an abstract view

Behavior space is denser than program space. → we can build a bigger map.

When do two programs behave the same?

▶ all programs that do nothing are equivalent.
▶ equivalent in every context ⇒ equivalent.
Program Behavior: an abstract view

Behavior space is denser than program space. 
→ we can build a bigger map.

When do two programs behave the same?

► all programs that do nothing are equivalent.

► equivalent in every context \(\Rightarrow\) equivalent.

→ defines a maximally dense space.
Program Behavior: an abstract view

Behavior space is denser than program space. 

→ we can build a bigger map.

When do two programs behave the same?

▶ all programs that do nothing are equivalent.

▶ equivalent in every context ⇒ equivalent.

→ defines a maximally dense space.

*this* is the space to map
Mapping the space of programs

where to start?
Mapping the space of programs

where to start?

Programs are just elements of an algebra
where to start?

Programs are just elements of an algebra

- a few constants
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)

- a binary operation (function application)
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)
- a binary operation (function application)
- a few equations

Try computational algebra methods: Todd-Coxeter algorithm builds a group
Generalize to non-associative algebra
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)
- a binary operation (function application)
- a few equations (or inequalities)

Try computational algebra methods: Todd-Coxeter algorithm builds a group
Generalize to non-associative algebra
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)
- a binary operation (function application)
- a few equations (or inequalities)

Try computational algebra methods:
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)
- a binary operation (function application)
- a few equations (or inequalities)

Try computational algebra methods:
  Todd-Coxeter algorithm builds a group
Mapping the space of programs

where to start?

Programs are just elements of an algebra

- a few constants
  (which constants = which language in family)
- a binary operation (function application)
- a few equations (or inequalities)

Try computational algebra methods:
  Todd-Coxeter algorithm builds a group
  Generalize to non-associative algebra
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S$, $K$, $J$

Then enlarge the map:

- choose two random programs
- apply one to the other (add row+column)
- enforce simple algebraic rules
- sometimes merging programs (slow)

When map gets too big, randomly prune programs.
How to make a map?

Todd-Coxeter-like algorithm:

- Start with basic programs, say $S, K, J$
- Then enlarge the map:
  - Choose two random programs
  - Apply one to the other (add row+column)
  - Enforce simple algebraic rules sometimes merging programs (slow)
- When map gets too big, randomly prune programs
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs,
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S, K, J$
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S, K, J$

Then enlarge the map:
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say S, K, J

Then enlarge the map:

- choose two random programs
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S, K, J$

Then enlarge the map:

- choose two random programs
- apply one to the other (add row+column)
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S, K, J$

Then enlarge the map:

- choose two random programs
- apply one to the other (add row+column)
- enforce simple algebraic rules
How to make a map?

**Todd-Coxeter-like algorithm:**

Start with basic programs, say $S, K, J$

Then enlarge the map:

- choose two random programs
- apply one to the other (add row+$\text{column}$)
- enforce simple algebraic rules
  sometimes merging programs
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S, K, J$

Then enlarge the map:

- choose two random programs
- apply one to the other (add row + column)
- enforce simple algebraic rules

sometimes merging programs (slow)
How to make a map?

Todd-Coxeter-like algorithm:

Start with basic programs, say $S, K, J$

Then enlarge the map:

- choose two random programs
- apply one to the other (add row+column)
- enforce simple algebraic rules
  sometimes merging programs (slow)

When map gets too big, randomly prune programs
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>??</td>
<td>??</td>
</tr>
<tr>
<td>K</td>
<td>??</td>
<td>??</td>
</tr>
</tbody>
</table>

\[ S \times y \times z = x \times z(y \times z) \]

\[ K \times x \times y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times y \ z = x \ z(y \ z) \]

\[ K \times y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
</tr>
</tbody>
</table>

\[
S \ x \ y \ z = x \ z(y \ z)
\]

\[
K \ x \ y = x
\]
Making a map: a simple example

<table>
<thead>
<tr>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[
S \times y \, z = x \, z(y \, z)
\]

\[
K \times y = x
\]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times y \times z = x \times z(y \times z) \]

\[ K \times y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>?</td>
<td>S</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times y z = x \times z(y \times z) \]

\[ K \times y = x \]
Making a map: a simple example

\[
S \quad K \quad KS
\]

\[
\begin{array}{ccc}
K & KS & ? & ? \\
KS & ? & S & ? \\
\end{array}
\]

\[
S \times y \ z = x \ z(y \ z)
\]

\[
K \times y = x
\]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times y \circ z = x \circ z(y \circ z) \]

\[ K \times y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
</tbody>
</table>

S x y z = x z(y z)

K x y = x
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

\[
S \times y z = x \circ z(y \circ z)
\]

\[
K \times y = x
\]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

\[ S x y z = x z(y z) \]
\[ K x y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

\[
S \times y \times z = x \times z(y \times z)
\]

\[
K \times x \times y = x
\]
Making a map: a simple example

\[
\begin{array}{c|ccc}
 & S & K & KS \\
\hline
S & ? & SK & ? \\
K & KS & ? & ? \\
KS & S & S & S \\
\end{array}
\]

\[
S \ x \ y \ z = x \ z(y \ z)
\]

\[
K \ x \ y = x
\]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

S x y z = x z(y z)

K x y = x
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>S x y z = x z(y z)</th>
<th>K x y = x</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>S x y z = x z(y z)</td>
<td>K x y = x</td>
</tr>
</tbody>
</table>

no rules apply
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[S \times y \times z = x \times z(y \times z)\]

\[K \times y = x\]
## Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times x y z = x \times z(y \times z) \]

\[ K \times x y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
<th>SKK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>SKK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times y z = x \times z(y \times z) \]

\[ K \times x y = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
<th>SKK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>SKK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S(x y z) = x (z(y z)) \]

\[ K(x y) = x \]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
<th>SKK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>SKK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

(SKK)(S) = (KS)(KS)

S x y z = x z(y z)

K x y = x
Making a map: a simple example

\[
\begin{array}{cccccc}
S & K & KS & SK & SKK \\
\end{array}
\]

\[(SKK)(S) = (KS)(KS)\]
\[S \times y \times z = x \times z(y \times z)\]
\[K \times y = x\]
Making a map: a simple example

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>K</th>
<th>KS</th>
<th>SK</th>
<th>SKK</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>?</td>
<td>SK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>K</td>
<td>KS</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>KS</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>?</td>
</tr>
<tr>
<td>SK</td>
<td>?</td>
<td>SKK</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>SKK</td>
<td>S</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

\[ S \times y \times z = \times z(y \times z) \]

\[ K \times y = y \]
Making a map: a large example
Making a map: a large example

Two data structures take most of the space
Making a map: a large example

1. a multiplication table
4868 x 4868 Application Table
8.6% Full = 2,036,584 Equations
Making a map: a large example

2. an order relation table
4868 x 4868 Order Relations
98.1% Decided + 1.9% Undecided
How much does it cost?

In Theory:

N programs,

N^2 space,

N^3 time,

equivalence is undecidable

In Practice:

12K programs,

1G Bytes,

1 month

equivalence is over 96% decided
How much does it cost?

**In Theory:**

$N$ programs, $N^2$ space,

**In Practice:**

12K programs, 1G Bytes, 1 month equivalence is over 96% decided
How much does it cost?

In Theory:

N programs, \( N^2 \) space, \( N^3 \) time,
How much does it cost?

In Theory:

$N$ programs, $N^2$ space, $N^3$ time,

equivalence is undecidable
How much does it cost?

In Theory:

N programs, \( N^2 \) space, \( N^3 \) time,
equivalence is undecidable

In Practice:

12K programs,
How much does it cost?

In Theory:

\[ N \text{ programs, } N^2 \text{ space, } N^3 \text{ time,} \]

equivalence is undecidable

In Practice:

12K programs, 1G Bytes,
How much does it cost?

In Theory:

N programs, \( N^2 \) space, \( N^3 \) time,

equivalence is undecidable

In Practice:

12K programs, 1G Bytes, 1 month
How much does it cost?

**In Theory:**

N programs, $N^2$ space, $N^3$ time,

equivalence is undecidable

**In Practice:**

12K programs, 1G Bytes, 1 month

equivalence is over 96% decided
A space of programs

What shape is the algebra of programs?
A space of programs

What shape is the algebra of programs?

- Program size gives a norm $|x|$
A space of programs

What shape is the algebra of programs?
  ▶ Program size gives a norm $|x|$
    (Kolmogorov complexity)
A space of programs

What shape is the algebra of programs?

- Program size gives a norm $|x|$
  (Kolmogorov complexity)

- also relative complexity $|x|_y$
A space of programs

What shape is the algebra of programs?

- Program size gives a norm $|x|$ (Kolmogorov complexity)
- also relative complexity $|x|_y$
- symmetrizing gives a distance $d(x, y) = |x|_y + |y|_x$

Gromov studied the geometry of groups this is a non-associative generalization
A space of programs

What shape is the algebra of programs?

- Program size gives a norm $|x|$
  (Kolmogorov complexity)

- also relative complexity $|x|_y$

- symmetrizing gives a distance
  $d(x, y) = |x|_y + |y|_x$

- this space is asymptotically hyperbolic:
  volume of sphere is exponential in radius
A space of programs

What shape is the algebra of programs?

- Program size gives a norm $|x|$ (Kolmogorov complexity)

- Also relative complexity $|x|_y$

- Symmetrizing gives a distance
  \[ d(x, y) = |x|_y + |y|_x \]

- This space is asymptotically hyperbolic: volume of sphere is exponential in radius

Gromov studied the geometry of groups
A space of programs

What shape is the algebra of programs?

- Program size gives a norm $|x|$
  (Kolmogorov complexity)

- also relative complexity $|x|_y$

- symmetrizing gives a distance
  $d(x, y) = |x|_y + |y|_x$

- this space is asymptotically hyperbolic:
  volume of sphere is exponential in radius

Gromov studied the geometry of groups
this is a non-associative generalization
Visualizing the space of programs

Goal Programming styles are local
Visualizing the space of programs

**Goal**  Programming styles are local
  - atoms are far-out
Visualizing the space of programs

Goal  Programming styles are local
    ▶ atoms are far-out
    ▶ related programs are close together
Visualizing the space of programs

**Goal**  Programming styles are local
  ▶ atoms are far-out
  ▶ related programs are close together
  ▶ parse trees are small

**How**  Pose as eigenvector problem
Visualizing the space of programs

**Goal**  Programming styles are local
  - atoms are far-out
  - related programs are close together
  - parse trees are small

**How**  Pose as eigenvector problem
  - linear springs between programs
Visualizing the space of programs

**Goal** Programming styles are local
- atoms are far-out
- related programs are close together
- parse trees are small

**How** Pose as eigenvector problem
- linear springs between programs
- simpler programs are heavier
Visualizing the space of programs

**Goal**  Programming styles are local
- atoms are far-out
- related programs are close together
- parse trees are small

**How**  Pose as eigenvector problem
- linear springs between programs
- simpler programs are heavier
- project to first few dimensions:
Visualizing the space of programs

Goal  Programming styles are local
  ▶ atoms are far-out
  ▶ related programs are close together
  ▶ parse trees are small

How  Pose as eigenvector problem
  ▶ linear springs between programs
  ▶ simpler programs are heavier
  ▶ project to first few dimensions: 3 space
Visualizing the space of programs

Goal  Programming styles are local
  ▶  atoms are far-out
  ▶  related programs are close together
  ▶  parse trees are small

How  Pose as eigenvector problem
  ▶  linear springs between programs
  ▶  simpler programs are heavier
  ▶  project to first few dimensions:
    3 space + 3 color
see interactive maps...

www.math.cmu.edu/~fho/johann/
Where to map?

Time complexity is cubic:
Where to map?

Time complexity is cubic: mis-fitting is expensive!
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis,
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis,

\[
\text{program} ::= \begin{array}{c}
S \\
K \\
J \\
(\text{program program})
\end{array}
\]
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis, with simple weights

\[
\text{program} ::= \begin{array}{c}
S @ 1/6 \\
| K @ 1/6 \\
| J @ 1/6 \\
| (\text{program program}) @ 1/2
\end{array}
\]
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis, with simple weights

\[
\text{program} ::= S @ 1/6 \\
\quad \mid K @ 1/6 \\
\quad \mid J @ 1/6 \\
\quad \mid (\text{program } \text{program}) @ 1/2
\]

Choice of small basis is arbitrary
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis, with simple weights

\[
\text{program} ::= \begin{align*}
S & \quad @ \frac{1}{6} \\
K & \quad @ \frac{1}{6} \\
J & \quad @ \frac{1}{6} \\
(\text{program} \ \text{program}) & \quad @ \frac{1}{2}
\end{align*}
\]

Choice of small basis is arbitrary
⇒ extra information
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis, with simple weights

\[
\text{program} ::= \begin{array}{c}
S @ 1/6 \\
| K @ 1/6 \\
| J @ 1/6 \\
| (\text{program} \ \text{program}) @ 1/2
\end{array}
\]

Choice of small basis is arbitrary
\implies \text{extra information} \implies \text{not simple}
Where to map?

Time complexity is cubic: mis-fitting is expensive!

A simple basis, with simple weights

\[
\text{program} ::= \begin{array}{l}
S @ 1/6 \\
| K @ 1/6 \\
| J @ 1/6 \\
| (\text{program program}) @ 1/2 
\end{array}
\]

Choice of small basis is arbitrary
\[\rightarrow\] extra information \[\rightarrow\] not simple

which languages are simple?
Complexity

Kolmogorov’s view: complexity is a norm
Complexity

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$
Complexity $\rightarrow$ Probability

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$

so consider...

a probability space of programs
Complexity → Probability

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$

so consider...

a probability space of programs
parametrized by a language
Complexity → Probability

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$

so consider...

a probability space of programs
parametrized by a language

($= \text{weighted set of basic programs}$)
Complexity → Probability → Geometry

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$

so consider...

a probability space of programs
parametrized by a language
(= weighted set of basic programs)

now we have...

a space of languages
Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$

so consider...

a probability space of programs
parametrized by a language
(= weighted set of basic programs)

now we have...

a space of languages: information manifold
Complexity → Probability → Geometry

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is $-\log(\text{probability})$

so consider...

a probability space of programs
parametrized by a language
($\equiv$ weighted set of basic programs)

now we have...

a space of languages: information manifold
\[\Rightarrow \text{Riemannian manifold}\]
Complexity → Probability → Geometry

Kolmogorov’s view: complexity is a norm
Solomonoff’s view: complexity is \(-\log(\text{probability})\)

so consider...

a probability space of programs parametrized by a language

\((\equiv \text{weighted set of basic programs})\)

now we have...

a space of languages: information manifold

\(\implies \text{Riemannian manifold}\)

\(\implies \text{differential manifold}\)
3-dimensional subspace

languages

simplex parametrizes probability distribution

entropy

S

J

K
inaccessible region
finite entropy region
parametrizes probability distribution
simplex
entropy
−(−)
S
J
K
1/2

entropy

simplex parametrizes probability distribution
So What?
How to find a simple language

**Goal:** map interesting programs
How to find a simple language

Goal: map interesting programs

Constraint: limited space and time
How to find a simple language

Goal: map interesting programs
Constraint: limited space and time

so find a language that makes interesting programs simple
How to find a simple language

Goal: map interesting programs
Constraint: limited space and time

so find a language that makes interesting programs simple

We know many interesting programs.
How to find a simple language

Goal: map interesting programs
Constraint: limited space and time

so find a language that makes interesting programs simple

We know many interesting programs. Language space is a Riemannian manifold.
How to find a simple language

Goal: map interesting programs
Constraint: limited space and time

so find a language that makes interesting programs simple

We know many interesting programs. Language space is a Riemannian manifold.

so collect a training set of programs
How to find a simple language

Goal: map interesting programs
Constraint: limited space and time

so find a language that makes interesting programs simple

We know many interesting programs.
Language space is a Riemannian manifold.

so collect a training set of programs, and do gradient descent to minimize its complexity
parametrizes probability distribution
simplex
entropy
$-(-)$
$\frac{1}{2}$
start
simplify

simplex parametrizes probability distribution

entropy
Potential Applications

Program simplification
Potential Applications

Program simplification
using database of simplest rewrites
Potential Applications

Program simplification using database of simplest rewrites

Software analysis
Potential Applications

Program simplification
using database of simplest rewrites

Software analysis
refactor based on spatial proximity
Potential Applications

Program simplification
  using database of simplest rewrites

Software analysis
  refactor based on spatial proximity

Universal Bayesian filtering
Potential Applications

Program simplification
using database of simplest rewrites

Software analysis
refactor based on spatial proximity

Universal Bayesian filtering
practical Solomonoff induction?
Potential Applications

Program simplification
using database of simplest rewrites

Software analysis
refactor based on spatial proximity

Universal Bayesian filtering
practical Solomonoff induction?

Programming by searching
Potential Applications

Program simplification
using database of simplest rewrites

Software analysis
refactor based on spatial proximity

Universal Bayesian filtering
practical Solomonoff induction?

Programming by searching
calibrate search with examples
Potential Applications

- Program simplification using database of simplest rewrites
- Software analysis refactor based on spatial proximity
- Universal Bayesian filtering practical Solomonoff induction
- Programming by searching calibrate search with examples
- Bayesian foundation for genetic programming
Potential Applications

Program simplification
  using database of simplest rewrites

Software analysis
  refactor based on spatial proximity

Universal Bayesian filtering
  practical Solomonoff induction?

Programming by searching
  calibrate search with examples
  Bayesian foundation for genetic programming
Summary

- The average over all languages is simpler than any one.
Summary

- the average over all languages is simpler than any one

- complexity $\rightarrow$ probability
Summary

- the average over all languages is simpler than any one

- complexity → probability → geometry
Summary

- the average over all languages is simpler than any one
- complexity → probability → geometry
- learn simplicity from examples
Summary

- the average over all languages is simpler than any one

- complexity → probability → geometry

- learn simplicity from examples

Questions

- what should those examples be?
Summary

- the average over all languages is simpler than any one
- complexity $\rightarrow$ probability $\rightarrow$ geometry
- learn simplicity from examples

Questions

- what should those examples be?
- how is real software shaped?