Using Kolmogorov Complexity to Make All the Art

Jonathan Langke and Peter Boothe

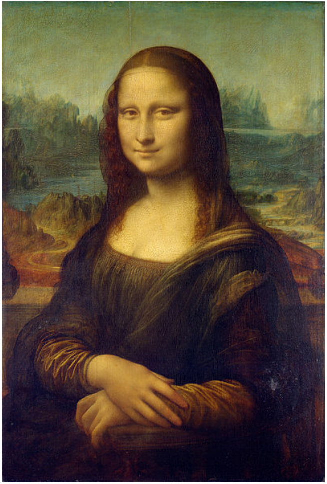
jlangke.student@manhattan.edu
Computer Engineering '13

peter.boothe@manhattan.edu
Computer Science

Manhattan College

6 April 2013

Art!



Art!

Art!

Art!

Int × Int → Bool

- A restricted form of digital art
- black and white pixels on a grid
- black is “true”, white is “false”
- each pixel’s color is a function of its grid coordinates
- art is a function mapping $(\mathbb{Z} \times \mathbb{Z})$ to $\{true, false\}$

Int × Int → Bool

- A restricted form of digital art
- black and white pixels on a grid
- black is "true", white is "false"
- each pixel's color is a function of its grid coordinates
- art is a function mapping $(\mathbb{Z} \times \mathbb{Z})$ to $\{true, false\}$

Some art is more complex than other art

Example

(x < y)

(((1 + (1 + 1)) < (x * y)) and ((x + 1) < (y * (1 + 1))))

Example

(x < y)

3 symbols

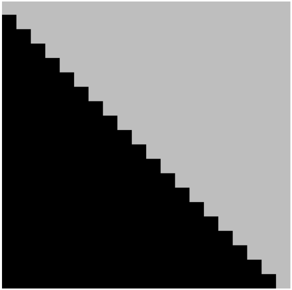
(((1 + (1 + 1)) < (x * y)) and ((x + 1) < (y * (1 + 1))))

19 symbols

Example

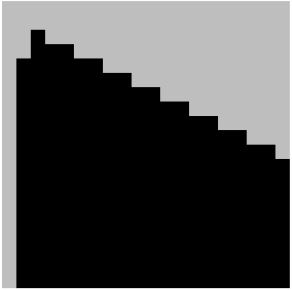
(x < y)

3 symbols



(((1 + (1 + 1)) < (x * y)) and ((x + 1) < (y * (1 + 1))))

19 symbols



($+x$ is to the right, $+y$ is down — it's a computer graphics thing)

Visual complexity

Visual complexity fuzzy and hard to define, but intuitive

Computational complexity well defined, but complicated and (arguably) unintuitive

Perhaps visual complexity is correlated with computational complexity?

Complexity, quantified

- The complexity of an object is the size of the smallest program (a.k.a. formula) which outputs that object.
- That's Kolmogorov Complexity! (KC)

Complexity, quantified

- The complexity of an object is the size of the smallest program (a.k.a. formula) which outputs that object.
- That's Kolmogorov Complexity! (KC)
- What programming language?

Complexity, quantified

- The complexity of an object is the size of the smallest program (a.k.a. formula) which outputs that object.
- That's Kolmogorov Complexity! (KC)
- What programming language?
  Doesn't matter, they are all equivalent!
  (The KC of an object in one language is equivalent to the KC of that same object in another language, up to an additive constant.)

Complexity, quantified

- The complexity of an object is the size of the smallest program (a.k.a. formula) which outputs that object.
- That's Kolmogorov Complexity! (KC)
- What programming language?
  Doesn't matter, they are all equivalent!
  (The KC of an object in one language is equivalent to the KC of that same object in another language, up to an additive constant.)
- What programming language?

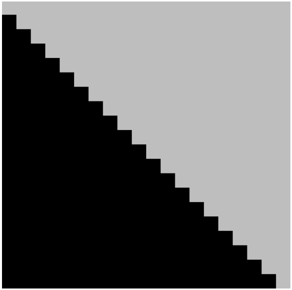
Complexity, quantified

- The complexity of an object is the size of the smallest program (a.k.a. formula) which outputs that object.
- That's Kolmogorov Complexity! (KC)
- What programming language?
  Doesn't matter, they are all equivalent!
  (The KC of an object in one language is equivalent to the KC of that same object in another language, up to an additive constant.)
- What programming language?
  Racket.

Example

(x < y)

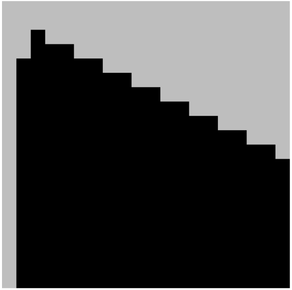
3 symbols



Kolmogorov complexity of 3

(((1 + (1 + 1)) < (x * y)) and ((x + 1) < (y * (1 + 1))))

19 symbols



Kolmogorov complexity of 19

(Remember: $+x$ is to the right, $+y$ is down)

PROgramming with LOGic

- Specify relationships
- Computer generates objects which satisfy those relationships
- We used MiniKanren, a library for Racket[1] developed for this purpose

[1] Racket is a descendant of Scheme and LISP

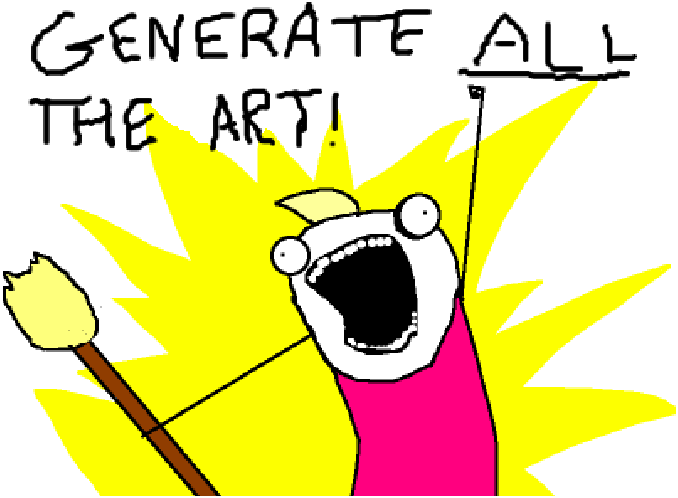
MiniKanren

- A logic programming language/library for Racket.
- Specify relationships, e.g.:

```
(def cousin
   (λ (p1 p2)
      (and (== (grandparent p1) (grandparent p2))
           (!= (parent p1) (parent p2)))))
```

- When asked appropriately, MiniKanren will generate objects which satisfy a relation. e.g.:

```
(run* (x) (cousin x 'JonathanLangke))
```



http://hyperboleandahalf.blogspot.com/2010/06/this-is-why-ill-never-be-adult.html

Our Algorithm

1. Define what it means to be a “well-typed” (internally consistent, i.e. non-crashing) program of a particular size
2. Generate, using MiniKanren, all well-typed programs up to a particular size
3. Run each program to generate its corresponding image
4. Record the smallest program which generates each image

Our Algorithm

1. Define what it means to be a "well-typed" (internally consistent, i.e. non-crashing) program of a particular size
2. Generate, using MiniKanren, all well-typed programs up to a particular size
3. Run each program to generate its corresponding image
4. Record the smallest program which generates each image

5. PROFIT!

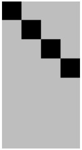
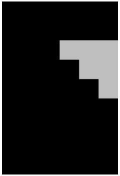
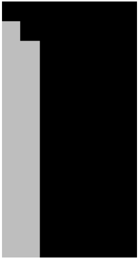
MANHATTAN COLLEGE

All the 2x2 Art

Formulae	Level	Pictures
none	0	empty
(true), (false)	1	
none	2	empty
(x < 1), (y < 1), (x < y), (0 < x), (0 < y), (y < x)	3	
(not (x < y)), (not (y < x))	4	
((y + x) < 1), ((y * x) < 1), (0 < (y + x)), (1 < (y + x))	5	
none	6	empty
((y < x) or (x < y))	7	
(not ((y < x) or (x < y)))	8	

(Remember: $+x$ is to the right, $+y$ is down)

Some Larger Art

Art					
Size	4x8	5x7	6x9	7x13	9x3
KC	7	7	6	6	5

References

William E. Byrd and Daniel P. Friedman.
From variadic functions to variadic relations.
In *Proceedings of the 2006 Scheme and Functional Programming Workshop*, 2006.

Daniel P Friedman, William E. Byrd, and Oleg Kiselyov.
The Reasoned Schemer.
MIT Press, 2005.

Danny Yoo.
`http://lists.racket-lang.org/users/archive/2006-December/015837.html`.

What We Did

1. Defined what it meant to be an acceptable program
    1. Built out of <, and, or, not, 0, 1, x, y, +, $*$
    2. Formula evaluates to true or false
2. Defined art
    1. Black and white pixel grid, corresponds to a boolean formula
    2. 
3. Used MiniKanren to generate, from our constraints in step 1, all formulae of a given size
4. Evaluated each formula (i.e. executed each program) to find its pictorial output.
    1. When an output was produced multiple times, we chose the smallest formula
5. Put it all together to discover the Kolmogorov Complexity of very small pieces of art.

What We Did

1. Defined what it meant to be an acceptable program
   1. Built out of $<$, and, or, not, 0, 1, x, y, +, $*$
   2. Formula evaluates to true or false
2. Defined art
   1. Black and white pixel grid, corresponds to a boolean formula
   2. 
3. Used MiniKanren to generate, from our constraints in step 1, all formulae of a given size
4. Evaluated each formula (i.e. executed each program) to find its pictorial output.
   1. When an output was produced multiple times, we chose the smallest formula
5. Put it all together to discover the Kolmogorov Complexity of very small pieces of art.

Any questions?