Kolmogorov Complexity

Eyad Loutfi

Section 1

Basics

Computability

- A program is something that takes in a string as input and has one as output, all within some model of computation – more on this in a bit.

Computability

- A program is something that takes in a string as input and has one as output, all within some model of computation – more on this in a bit.
- A natural question to ask is if there is a correspondence between strings and the programs that output them.

Computability

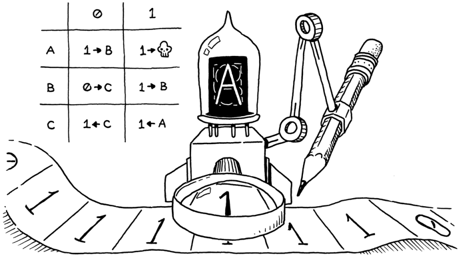
- A program is something that takes in a string as input and has one as output, all within some model of computation – more on this in a bit.
- A natural question to ask is if there is a correspondence between strings and the programs that output them.
- To what extent might a "large" string be produced by a "short" program, i.e. how much might we be able to "compress" strings?

Turing Machines

- First we need to introduce models of computation. The classic model of computation is the *Turing machine*.

Turing Machines

- First we need to introduce models of computation. The classic model of computation is the *Turing machine*.
- TM's are defined over a finite alphabet, have an infinite tape that has the input at the start of it, a read head with internal state that starts at the start of the tape and can write or move left or right all according to some finite instructions (transition function based on the head's state and character in the cell it's over).



Computability

- This turns out to be as powerful as any computer we know! The *Church Turing* thesis states that anything a computer can do can be done by a Turing machine.

Computability

- This turns out to be as powerful as any computer we know! The *Church Turing* thesis states that anything a computer can do can be done by a Turing machine.
- When we say model of computation, we mean anything that can represent all the functions a computer can, which all your favorite programming languages (C, Python) can do since they're Turing Complete (can do anything a Turing machine can).

Computability

- This turns out to be as powerful as any computer we know! The *Church Turing* thesis states that anything a computer can do can be done by a Turing machine.
- When we say model of computation, we mean anything that can represent all the functions a computer can, which all your favorite programming languages (C, Python) can do since they're Turing Complete (can do anything a Turing machine can).
- Note that programs/TM's etc are defined by precise descriptions, so in a sense they are strings themselves!

Computability

- This turns out to be as powerful as any computer we know! The *Church Turing* thesis states that anything a computer can do can be done by a Turing machine.
- When we say model of computation, we mean anything that can represent all the functions a computer can, which all your favorite programming languages (C, Python) can do since they're Turing Complete (can do anything a Turing machine can).
- Note that programs/TM's etc are defined by precise descriptions, so in a sense they are strings themselves!
- One can devise an encoding scheme of all programs – many such encodings exist, could use all binary strings or all ASCII strings, etc.

Computability

- More importantly, the encoding is done in such a way that a program can recognize it – such that we can define a program to simulate the behavior of a machine as a result of parsing the encoding.

Computability

- More importantly, the encoding is done in such a way that a program can recognize it – such that we can define a program to simulate the behavior of a machine as a result of parsing the encoding.
- This means we can define a program to be able to simulate any other program – a very important result!

Computability

- More importantly, the encoding is done in such a way that a program can recognize it – such that we can define a program to simulate the behavior of a machine as a result of parsing the encoding.
- This means we can define a program to be able to simulate any other program – a very important result!
- Other important definitions: A set is "recursively enumerable," or RE if there exists a program that can give a "yes" answer to any string in the set. A set is "computable" if a program like that exists that also gives a "no" answer for any string not in the set.

Computability

- More importantly, the encoding is done in such a way that a program can recognize it – such that we can define a program to simulate the behavior of a machine as a result of parsing the encoding.
- This means we can define a program to be able to simulate any other program – a very important result!
- Other important definitions: A set is "recursively enumerable," or RE if there exists a program that can give a "yes" answer to any string in the set. A set is "computable" if a program like that exists that also gives a "no" answer for any string not in the set.
- These properties are not guaranteed in general due to possibility of infinite looping.

Kolmogorov Complexity

- Intuitively, 057835292 has the same probability of being selected as 000000000, yet one of these feels more "complex" to us.

Kolmogorov Complexity

- Intuitively, 057835292 has the same probability of being selected as 000000000, yet one of these feels more "complex" to us.
- Solution: We define the complexity of a string be the length of the shortest program that prints it. This is *Kolmogorov complexity*.

Kolmogorov Complexity

- Intuitively, 057835292 has the same probability of being selected as 000000000, yet one of these feels more “complex” to us.
- Solution: We define the complexity of a string be the length of the shortest program that prints it. This is *Kolmogorov complexity*.
- Limitation: Must be relative to a model of computation / programming language you fix.

Kolmogorov Complexity

- Intuitively, 057835292 has the same probability of being selected as 000000000, yet one of these feels more "complex" to us.
- Solution: We define the complexity of a string be the length of the shortest program that prints it. This is *Kolmogorov complexity*.
- Limitation: Must be relative to a model of computation / programming language you fix.
- Gives us a powerful tool for understanding compression, the distribution of "complexity" among finite strings themselves, and even limitative results of mathematics as a whole.

Section 2

Some Fundamental Proofs

Invariance Theorem

- It might seem like we are completely limited talking about Kolmogorov complexity across languages since it is relative to them.

Invariance Theorem

- It might seem like we are completely limited talking about Kolmogorov complexity across languages since it is relative to them.
- Rather, the K complexity only differs by a constant factor between different languages.

Invariance Theorem

Theorem

Given 2 languages L_1, L_2 and their respective K complexities K_1, K_2, for any string s, $K_1(s) \leq K_2(s) + c$ for some constant c.

Invariance Theorem

Theorem

Given 2 languages L_1, L_2 and their respective K complexities K_1, K_2, for any string s, $K_1(s) \leq K_2(s) + c$ for some constant c.

Proof

1. Suppose we have an interpreter in L_1 for strings p representing programs in L_2, $Interpret(p)$, meaning it will produce the output that p would have produced in L_2.

Invariance Theorem

Theorem

Given 2 languages L_1, L_2 and their respective K complexities K_1, K_2, for any string s, $K_1(s) \leq K_2(s) + c$ for some constant c.

Proof

1. Suppose we have an interpreter in L_1 for strings p representing programs in L_2, $Interpret(p)$, meaning it will produce the output that p would have produced in L_2.
2. Let P be the shortest program in L_2 that outputs s.

Invariance Theorem

Theorem

Given 2 languages L_1, L_2 and their respective K complexities K_1, K_2, for any string s, $K_1(s) \leq K_2(s) + c$ for some constant c.

Proof

1. Suppose we have an interpreter in L_1 for strings p representing programs in L_2, $Interpret(p)$, meaning it will produce the output that p would have produced in L_2.
2. Let P be the shortest program in L_2 that outputs s.
3. Then $Interpret(P)$ will produce s in L_1.

Invariance Theorem

Theorem

Given 2 languages L_1, L_2 and their respective K complexities K_1, K_2, for any string s, $K_1(s) \leq K_2(s) + c$ for some constant c.

Proof

1. Suppose we have an interpreter in L_1 for strings p representing programs in L_2, $Interpret(p)$, meaning it will produce the output that p would have produced in L_2.
2. Let P be the shortest program in L_2 that outputs s.
3. Then $Interpret(P)$ will produce s in L_1.
4. We know $|Interpret(P)| = |Interpret| + |P|$, but $|P| = K_2(s)$ and we can consider $|Interpret|$ to be some constant c.

Invariance Theorem

Theorem

Given 2 languages L_1, L_2 and their respective K complexities K_1, K_2, for any string s, $K_1(s) \leq K_2(s) + c$ for some constant c.

Proof

1. Suppose we have an interpreter in L_1 for strings p representing programs in L_2, $Interpret(p)$, meaning it will produce the output that p would have produced in L_2.
2. Let P be the shortest program in L_2 that outputs s.
3. Then $Interpret(P)$ will produce s in L_1.
4. We know $|Interpret(P)| = |Interpret| + |P|$, but $|P| = K_2(s)$ and we can consider $|Interpret|$ to be some constant c.
5. Therefore $K_1(s) \leq K_2(s) + c$.

Universal Lossless Compression

Theorem

For every n, there exists a length n string a whose Kolmogorov complexity $K(a)$ is at least n.

Universal Lossless Compression

Theorem

For every n, there exists a length n string a whose Kolmogorov complexity $K(a)$ is at least n.

Proof

1. Suppose for contradiction that every string a in $\{0,1\}^n$, of which there are 2^n of, $K(a) < n$.

Universal Lossless Compression

Theorem

For every n, there exists a length n string a whose Kolmogorov complexity $K(a)$ is at least n.

Proof

1. Suppose for contradiction that every string a in $\{0,1\}^n$, of which there are 2^n of, $K(a) < n$.
2. Then we have a one-to-one function f, such that $f(a)$ will output the string representing the shortest program that outputs a.

Universal Lossless Compression

Theorem

For every n, there exists a length n string a whose Kolmogorov complexity $K(a)$ is at least n.

Proof

1. Suppose for contradiction that every string a in $\{0,1\}^n$, of which there are 2^n of, $K(a) < n$.
2. Then we have a one-to-one function f, such that $f(a)$ will output the string representing the shortest program that outputs a.
3. Number of shorter strings is $\sum_{i=0}^{n-1} \left|\{0,1\}^i\right| = 1 + 2 + 2^2 + ... + 2^{n-1} = 2^n - 1$

Universal Lossless Compression

Theorem

For every n, there exists a length n string a whose Kolmogorov complexity $K(a)$ is at least n.

Proof

1. Suppose for contradiction that every string a in $\{0, 1\}^n$, of which there are 2^n of, $K(a) < n$.
2. Then we have a one-to-one function f, such that $f(a)$ will output the string representing the shortest program that outputs a.
3. Number of shorter strings is $\sum_{i=0}^{n-1} \left|\{0, 1\}^i\right| = 1 + 2 + 2^2 + ... + 2^{n-1} = 2^n - 1$
4. For f to be one-to-one, there would have be $\geq 2^n$ output programs but we only have $2^n - 1$. Contradiction

Universal Lossless Compression

- What this tells us is there will always be incompressible strings where $K(a) \geq |a|$, i.e. universal lossless compression is impossible!

Universal Lossless Compression

- What this tells us is there will always be incompressible strings where $K(a) \geq |a|$, i.e. universal lossless compression is impossible!
- We can now prove an even more general fact regarding the computability of K!

Uncomputability of K

The proof uses the fact programs can simulate other programs and that there is a binary string encoding scheme for all programs

Theorem

There is no possible program $P(x, i)$ that outputs "yes" if $K(x) = i$ and "no" if $K(x) \neq i$.

Uncomputability of K

The proof uses the fact programs can simulate other programs and that there is a binary string encoding scheme for all programs

Theorem

There is no possible program $P(x, i)$ that outputs "yes" if $K(x) = i$ and "no" if $K(x) \neq i$.

Proof

- Suppose there is such a program P. We construct another program Q(j) as follows:

Uncomputability of K

The proof uses the fact programs can simulate other programs and that there is a binary string encoding scheme for all programs

Theorem

There is no possible program $P(x, i)$ that outputs "yes" if $K(x) = i$ and "no" if $K(x) \neq i$.

Proof

- Suppose there is such a program P. We construct another program Q(j) as follows:

```
Q(j):
  for each binary string encoding x of a program
      Simulate P(x, j)
      If simulation outputs "yes"
          print x    << This will be the first program x s.t. K(x) = j >>
```

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.
- Therefore $P(y, j)$ outputs “yes”, which implies $K(y) = j$

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.
- Therefore $P(y, j)$ outputs "yes", which implies $K(y) = j$
- Since $Q(j)$ is a program that outputs y, the length of $Q(j)$ must be at least j.

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.
- Therefore $P(y, j)$ outputs "yes", which implies $K(y) = j$
- Since $Q(j)$ is a program that outputs y, the length of $Q(j)$ must be at least j.
- We know $Q(j)$ includes P in it's definition, in addition to needing to write the input j, so $|Q(j)| = |P| + |j| + c$ where c is some constant overhead by P.

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.
- Therefore $P(y, j)$ outputs "yes", which implies $K(y) = j$
- Since $Q(j)$ is a program that outputs y, the length of $Q(j)$ must be at least j.
- We know $Q(j)$ includes P in it's definition, in addition to needing to write the input j, so $|Q(j)| = |P| + |j| + c$ where c is some constant overhead by P.
- We need at most $\log(j)$ bits to write j, so we obtain the inequality $K(y) = j \leq |Q(j)| \leq |P| + \log(j) + c \Rightarrow j - \log(j) \leq |P| + c$

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.
- Therefore $P(y, j)$ outputs "yes", which implies $K(y) = j$
- Since $Q(j)$ is a program that outputs y, the length of $Q(j)$ must be at least j.
- We know $Q(j)$ includes P in it's definition, in addition to needing to write the input j, so $|Q(j)| = |P| + |j| + c$ where c is some constant overhead by P.
- We need at most $\log(j)$ bits to write j, so we obtain the inequality $K(y) = j \leq |Q(j)| \leq |P| + \log(j) + c \Rightarrow j - \log(j) \leq |P| + c$
- This will be true no matter what j we pick, yet since $j - \log(j)$ is unbounded and P and c are fixed, there will be a j that makes it greater than $|P| + c$, a contradiction.

Uncomputability of K Continued

Proof

- Let y be the output of $Q(j)$.
- Therefore $P(y, j)$ outputs "yes", which implies $K(y) = j$
- Since $Q(j)$ is a program that outputs y, the length of $Q(j)$ must be at least j.
- We know $Q(j)$ includes P in it's definition, in addition to needing to write the input j, so $|Q(j)| = |P| + |j| + c$ where c is some constant overhead by P.
- We need at most $\log(j)$ bits to write j, so we obtain the inequality $K(y) = j \leq |Q(j)| \leq |P| + \log(j) + c \Rightarrow j - \log(j) \leq |P| + c$
- This will be true no matter what j we pick, yet since $j - \log(j)$ is unbounded and P and c are fixed, there will be a j that makes it greater than $|P| + c$, a contradiction.
- Therefore Kolmogorov complexity is not computable, there cannot exist such a program P.

Section 3

What we can say about the complexity of strings?

Average Complexity of Finite Strings

- A natural question to ask is what the distribution of Kolmogorov complexity over the average binary string really looks like

Average Complexity of Finite Strings

- A natural question to ask is what the distribution of Kolmogorov complexity over the average binary string really looks like
- Intuitively we would expect most strings to appear random, and it turns out this intution is true: The vast majority of strings are not very compressible.

Average Complexity of Finite Strings

Theorem

If A is the set of all strings a where $K(a) < n - k$, then $\frac{|A|}{|\{0,1\}^n|} < \frac{1}{2^k}$

Average Complexity of Finite Strings

Theorem

If A is the set of all strings a where $K(a) < n - k$, then $\frac{|A|}{|\{0,1\}^n|} < \frac{1}{2^k}$

Proof

1. There is a one-to-one function from A to the set C of all binary strings with length less than $n - k$.
   - $|C| = 2^{n-k} - 1$

Average Complexity of Finite Strings

Theorem

If A is the set of all strings a where $K(a) < n - k$, then $\frac{|A|}{|\{0,1\}^n|} < \frac{1}{2^k}$

Proof

1. There is a one-to-one function from A to the set C of all binary strings with length less than $n - k$.
   - $|C| = 2^{n-k} - 1$
2. So $|A| \leq |C| = 2^{n-k} - 1 \Rightarrow |A| < 2^{n-k}$.

Average Complexity of Finite Strings

Theorem

If A is the set of all strings a where $K(a) < n - k$, then $\frac{|A|}{|\{0,1\}^n|} < \frac{1}{2^k}$

Proof

1. There is a one-to-one function from A to the set C of all binary strings with length less than $n - k$.
   - $|C| = 2^{n-k} - 1$
2. So $|A| \leq |C| = 2^{n-k} - 1 \Rightarrow |A| < 2^{n-k}$.
3. Therefore $\frac{|A|}{|\{0,1\}^n|} < \frac{2^{n-k}}{2^n} = \frac{1}{2^k}$.

Average Complexity of Finite Strings

- This implies the proportion of binary strings that can be compressed to be 2 bits shorter is less than a half, the ones that can be compressed to be 3 bits shorter is less than a quarter.

Average Complexity of Finite Strings

- This implies the proportion of binary strings that can be compressed to be 2 bits shorter is less than a half, the ones that can be compressed to be 3 bits shorter is less than a quarter.
- When looking at strings of length 100, the proportion of them that can be compressed to be even just 10 bits less is less than $\frac{1}{2^9}$, or less than about 0.2%

Average Complexity of Finite Strings

- This implies the proportion of binary strings that can be compressed to be 2 bits shorter is less than a half, the ones that can be compressed to be 3 bits shorter is less than a quarter.
- When looking at strings of length 100, the proportion of them that can be compressed to be even just 10 bits less is less than $\frac{1}{2^9}$, or less than about 0.2%
- Not only are most strings hard to compress, but we can actually prove an upper bound on the Kolmogorov complexity that we are even able to PROVE a string to have.

Proof System

- Before we continue, we'll introduce the notion of a *proof system*.

Proof System

- Before we continue, we'll introduce the notion of a *proof system*.
- A proof system is defined via
  - A computable language (set of strings) which we'll usually take to be well formed formulas expressed in some logic

Proof System

- Before we continue, we'll introduce the notion of a *proof system*.
- A proof system is defined via
  - A computable language (set of strings) which we'll usually take to be well formed formulas expressed in some logic
  - A distinguished RE subset of that language we call the *axioms*
  - A set of *theorems* that include the axioms.

Proof System

- Before we continue, we'll introduce the notion of a *proof system*.
- A proof system is defined via
  - A computable language (set of strings) which we'll usually take to be well formed formulas expressed in some logic
  - A distinguished RE subset of that language we call the *axioms*
  - A set of *theorems* that include the axioms.
  - A set of inference rules that can be programmed and can be applied to any theorem to create other theorems.

The set of all theorems that can be produced this way is then called the proof system.

Proof System

- Before we continue, we'll introduce the notion of a *proof system*.
- A proof system is defined via
  - A computable language (set of strings) which we'll usually take to be well formed formulas expressed in some logic
  - A distinguished RE subset of that language we call the *axioms*
  - A set of *theorems* that include the axioms.
  - A set of inference rules that can be programmed and can be applied to any theorem to create other theorems.

  The set of all theorems that can be produced this way is then called the proof system.
- Intuitively this is just a model of how we prove things, but the key insight is we only ever do proofs using computable inference rules.

Example Inference Rules in First Order Logic

- Clearly proof systems are RE – one can just make a program that starts at the axioms and repeatedly apply the inference rules – like these from first order logic.

- **Propositional Tautologies:** A or not A, not(A and not A), etc. are valid.
- **Modus Ponens:** If A is valid and A implies B is valid then B is valid.
- **Equality Rules:** x=x, x=y implies y=x, x=y and y=z implies x=z, and x=y implies f(x)=f(y) are all valid.
- **Change of Variables:** Changing variable names leaves a statement valid.
- **Quantifier Elimination:** If For all x, A(x) is valid, then A(y) is valid for any y.
- **Quantifier Addition:** If A(y) is valid where y is an unrestricted variable, then For all x, A(x) is valid.
- **Quantifier Rules:** If Not(For all x, A(x)) is valid, then There exists an x such that Not(A(x)) is valid. Etc.

Math is Incomplete

- One result that immediately follows from the uncomputability of K is Godel's First Incompleteness theorem, that any RE proof system for mathematics will not be able prove or disprove every statement in the language associated with it.
- This is because if we could, then the fact the proof system is RE implies we can have a program loop through all theorems till it finds the statement for what the Kolmogorov complexity of a string is or isn't for any string, which would make K computable which we already proved is impossible.

Math is Incomplete

- One result that immediately follows from the uncomputability of K is Godel's First Incompleteness theorem, that any RE proof system for mathematics will not be able prove or disprove every statement in the language associated with it.
- This is because if we could, then the fact the proof system is RE implies we can have a program loop through all theorems till it finds the statement for what the Kolmogorov complexity of a string is or isn't for any string, which would make K computable which we already proved is impossible.
- Godel's result was more powerful, applying to any system that can prove facts regarding basic arithmetic, though this still applies since logical statements about programs may be encoded as arithmetical statements – i.e. Godel numbering.

Godel Numbering

- Fundamental Theorem of Arithmetic ensures that any Godel number has a unique prime factorization - allowing one to retrieve the original statement using the prime factors.

English	For all	numbers x	there exists	an immediate successor to	x
Math	∀	x	∃	s	x
Gödel number (for symbol)	9	11	4	7	11
Gödel numbering scheme	2^9	3^{11}	5^4	7^7	11^{11}
Result	512	177,147	625	823,543	285,311,670,611
Gödel number (for statement)	13,319,551,367,882,900,000,000,000,000				

Chaitin's Incompleteness Theorem

"The proof of this closely resembles G. G. Berry's paradox of 'the first natural number which cannot be named in less than a billion words.' The version of Berry's paradox that will do the trick is 'that object having the shortest proof that its algorithmic information content is greater than a billion bits." – Gregory Chaitin

Chaitin's Incompleteness Theorem

Theorem

For any proof system F, there is a largest number L such that F cannot prove that the Kolmogorov complexity of any bit string is more than L.

Chaitin's Incompleteness Theorem

Theorem

For any proof system F, there is a largest number L such that F cannot prove that the Kolmogorov complexity of any bit string is more than L.

- Suppose not, so there is some RE system F such that for any L, F can prove $K(x) = L$ for some x.

Chaitin's Incompleteness Theorem

Theorem

For any proof system F, there is a largest number L such that F cannot prove that the Kolmogorov complexity of any bit string is more than L.

- Suppose not, so there is some RE system F such that for any L, F can prove $K(x) = L$ for some x.
- We construct a program P that takes in an integer L and iterates through all proofs in F until it finds a proof of $K(x) = L$ for some x, then prints that x.

Chaitin's Incompleteness Theorem Continued

- Let x be the output of P for a certain L, therefore $K(x) = L$, and since $P(L)$ outputs x, $L \leq P(L) = |P| + |L| + c \leq |P| + \log(L) + c$ where c is some constant overhead by P.

Chaitin's Incompleteness Theorem Continued

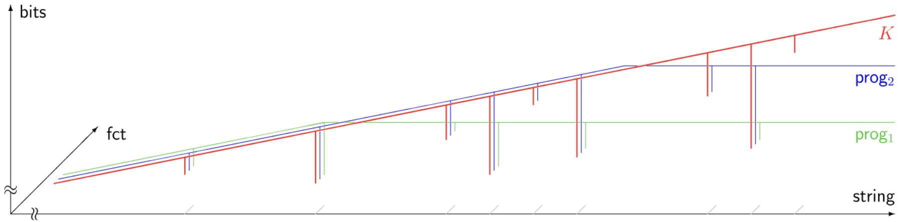
- Let x be the output of P for a certain L, therefore $K(x) = L$, and since $P(L)$ outputs x, $L \leq P(L) = |P| + |L| + c \leq |P| + \log(L) + c$ where c is some constant overhead by P.
- This means regardless of L, $L - \log(L) \leq |P| + c$, but since P and c are fixed, for large enough L this will clearly not hold.

Chaitin's Incompleteness Theorem Continued

- Let x be the output of P for a certain L, therefore $K(x) = L$, and since $P(L)$ outputs x, $L \leq P(L) = |P| + |L| + c \leq |P| + \log(L) + c$ where c is some constant overhead by P.
- This means regardless of L, $L - \log(L) \leq |P| + c$, but since P and c are fixed, for large enough L this will clearly not hold.
- Therefore for any such proof system F, there must exist an L such that no string can be proven to have Kolmogorov complexity of L, or any number bigger since it will still violate the inequality.

Chaitin's Incompleteness Theorem Continued

- This further implies any program that would take in an input and calculate a lower bound on it's Kolmogorov complexity cannot exceed some limit.



The End


HOW DO I GET TO YOUR PLACE FROM LEXINGTON?
HMM...
OK, STARTING FROM YOUR DRIVEWAY, TAKE EVERY LEFT THAT DOESN'T PUT YOU ON A PRIME-NUMBERED HIGHWAY OR STREET NAMED FOR A PRESIDENT.
WHEN PEOPLE ASK FOR STEP-BY-STEP DIRECTIONS, I WORRY THAT THERE WILL BE TOO MANY STEPS TO REMEMBER, SO I TRY TO PUT THEM IN MINIMAL FORM.