THE POWER OF ABSTRACTION IN EDUCATION

Arnold L. Rosenberg

University of Massachusetts
Amherst, MA, USA

Give a person a fish

Give a person a fish

—and you feed the person for a day

Give a person a fish

—and you feed the person for a day

Teach the person how to fish

Give a person a fish

—and you feed the person for a day

Teach the person how to fish

—*and you feed the person for life*

Give a person a fish

—and you feed the person for a day

Teach the person how to fish

—*and you feed the person for life*

Facts are like fish

Give a person a fish

—and you feed the person for a day

Teach the person how to fish

—*and you feed the person for life*

Facts are like fish

—Abstractions are like fishing equipment

Abstractions are like fishing equipment

When you are allowed to deal with abstractions, not objects

then you can devise *operational names* for the objects

—names that give you *control* over the objects

Abstractions are like fishing equipment

When you are allowed to deal with abstractions, not objects

then you can devise *operational names* for the objects

—names that give you *control* over the objects

And—

You can create different names for different purposes!

What's in a name? (Shakespeare)

What's in a name? (Shakespeare)

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

2. The second requires some computing/programming experience

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

2. The second requires some computing/programming experience

   We try to appeal to intuition, via "pictures"

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

2. The second requires some computing/programming experience

   We try to appeal to intuition, via "pictures"

3. The third is rather "heavy"

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

2. The second requires some computing/programming experience

   We try to appeal to intuition, via “pictures”

3. The third is rather “heavy”

   It involves a blend of philosophy and linguistic and mathematics

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

2. The second requires some computing/programming experience

   We try to appeal to intuition, via "pictures"

3. The third is rather "heavy"

   It involves a blend of philosophy and linguistic and mathematics

   I try to impress you by citing famous names . . .

You can devise *operational names* for the objects

—names that give you *control* over the objects

—different names for different purposes

We discuss three abstraction-naming situations

1. The first is rather elementary (in required background)

   We discuss this one in detail

2. The second requires some computing/programming experience

   We try to appeal to intuition, via “pictures”

3. The third is rather “heavy”

   It involves a blend of philosophy and linguistic and mathematics

   I try to impress you by citing famous names . . .

EXAMPLES ARE SIMPLIFIED TO APPEAL TO EVEN BEGINNING STUDENTS

Situation #1

HOW TO THINK ABOUT *NUMBERS*

Situation #1

HOW TO THINK ABOUT *NUMBERS*

The main message:

Try to think about numbers using a variety of "names".

Different "names" may suggest very different ways of thinking — and reasoning

Consider the following table.

$$\begin{array}{lcrcl}
1 & = & 1 & = & 1^2 \\
1+3 & = & 4 & = & 2^2 \\
1+3+5 & = & 9 & = & 3^2 \\
1+3+5+7 & = & 16 & = & 4^2 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Consider the following table.

$$\begin{array}{lcrcl}
1 & = & 1 & = & 1^2 \\
1+3 & = & 4 & = & 2^2 \\
1+3+5 & = & 9 & = & 3^2 \\
1+3+5+7 & = & 16 & = & 4^2 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

It *appears* that

each perfect square n^2 *is the sum of the first* n *odd numbers.*

Consider the following table.

$$\begin{array}{lcrcl}
1 & = & 1 & = & 1^2 \\
1+3 & = & 4 & = & 2^2 \\
1+3+5 & = & 9 & = & 3^2 \\
1+3+5+7 & = & 16 & = & 4^2 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

It *appears* that

each perfect square n^2 *is the sum of the first* n *odd numbers.*

Let's *prove* that this is true for all positive integers n.

Each perfect square n^2 is the sum of the first n odd numbers.

We provide *three* proofs which use different “names” for the numbers.

Each perfect square n^2 is the sum of the first n odd numbers.

We provide *three* proofs which use different “names” for the numbers.

(We could provide more “names” and more proofs.)

Each perfect square n^2 is the sum of the first n odd numbers.

We provide *three* proofs which use different “names” for the numbers.

Some “names” for the numbers:

$Numeral$ $(Digit)$	$Token$ $(Number)$
1	●
2	●●
3	● ● ●
4	● ● ●●
5	● ● ● ● ●
6	● ● ● ● ●●
7	● ● ● ● ● ● ●

Each perfect square n^2 is the sum of the first n odd numbers.

We provide *three* proofs which use different “names” for the numbers.

More “names” for the numbers:

$Numeral$ $(Digit)$	$Token$ $(Number)$	$Token$ $(Number + configuration)$
1	●	●
2	●●	
3	● ● ●	● ● ●
4	● ● ●●	
5	● ● ● ● ●	● ● ● ● ●
6	● ● ● ● ●●	
7	● ● ● ● ● ● ●	● ● ● ● ● ●

A TEXT-ORIENTED PROOF

Each perfect square n^2 is the sum of the first n odd numbers.

A *text-oriented* proof: STEP 1

Write out the summation of interest:

$$S = 1 + 3 + 5 + 7 + 9$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *text-oriented* proof: STEP 2

Rewrite the summation — *in reverse order*.

$$\begin{array}{cccccccccccc} S & = & 1 & + & 3 & + & 5 & + & 7 & + & 9 \\ S & = & 9 & + & 7 & + & 5 & + & 3 & + & 1 \end{array}$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *text-oriented* proof: STEP 3

Add the two versions of the summation.

$$\begin{array}{rcccccccccc} S & = & 1 & + & 3 & + & 5 & + & 7 & + & 9 \\ S & = & 9 & + & 7 & + & 5 & + & 3 & + & 1 \\ \hline 2S & = & 10 & + & 10 & + & 10 & + & 10 & + & 10 \end{array}$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *text-oriented* proof: STEP 3

Add the two versions of the summation.

$$\begin{array}{rcccccccccc} S & = & 1 & + & 3 & + & 5 & + & 7 & + & 9 \\ S & = & 9 & + & 7 & + & 5 & + & 3 & + & 1 \\ \hline 2S & = & 10 & + & 10 & + & 10 & + & 10 & + & 10 \end{array}$$

Note that all column-sums are identical.

Each perfect square n^2 is the sum of the first n odd numbers.

A *text-oriented* proof: STEP 3

Add the two versions of the summation.

$$\begin{array}{rcccccccccc} S & = & 1 & + & 3 & + & 5 & + & 7 & + & 9 \\ S & = & 9 & + & 7 & + & 5 & + & 3 & + & 1 \\ \hline 2S & = & 10 & + & 10 & + & 10 & + & 10 & + & 10 \end{array}$$

Note that all column-sums are identical.

We thereby find that

$$2S \quad = \quad 5 \times 10 \quad = \quad 50$$

so that

$$S \quad = \quad 25 \quad = \quad 5^2$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *text-oriented* proof: STEP 3

Add the two versions of the summation.

$$\begin{array}{rcccccccccc} S & = & 1 & + & 3 & + & 5 & + & 7 & + & 9 \\ S & = & 9 & + & 7 & + & 5 & + & 3 & + & 1 \\ \hline 2S & = & 10 & + & 10 & + & 10 & + & 10 & + & 10 \end{array}$$

Note that all column-sums are identical.

We thereby find that

$$2S \quad = \quad 5 \times 10 \quad = \quad 50$$

so that

$$S \quad = \quad 25 \quad = \quad 5^2$$

Similar calculations work for any n.

A TOKEN-NUMBER-ORIENTED PROOF

<u>Each perfect square n^2 is the sum of the first n odd numbers.</u>

A *<u>token-number-oriented</u>* proof: STEP 1

Write out the summation of interest:

$$S = 1 + 3 + 5 + 7 + 9 = \begin{array}{ll} & \bullet \\ + & \bullet \ \bullet \ \bullet \\ + & \bullet \ \bullet \ \bullet \ \bullet \ \bullet \\ + & \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \\ + & \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \end{array}$$

<u>*Each perfect square n^2 is the sum of the first n odd numbers.*</u>

A <u>*token-number-oriented*</u> proof: STEP 1

Write out the summation of interest:

$$
\begin{array}{rcl}
 & & \quad \bullet \\
 & & + \ \bullet \ \bullet \ \bullet \\
S & = & + \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \\
 & & + \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \\
 & & + \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet \ \bullet
\end{array}
$$

The *token-based* form of the summation will expose the advantage of this way of “naming” numbers.

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number-oriented* proof: STEP 3

The *token-based* form of the summation provides the following value for S:

$$S \quad = \quad \begin{array}{lllllllll} \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

S thus equals *the area of the height-*5*, base-*9 *right triangle.*

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number-oriented* proof: STEP 3

The *token-based* form of the summation provides the following value for S:

$$S \quad = \quad \begin{array}{lllllllll} \bullet & & & & & & & & \\ \bullet & \bullet & \bullet & & & & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

S thus equals *the area of the height-5, base-9 right triangle.*

Since the base and height of the triangle are both *odd*,
the area is not easy to calculate visually.

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number-oriented* proof: STEP 3

The *token-based* form of the summation provides the following value for S:

$$S \quad = \quad \begin{array}{lllllllll} \bullet & & & & & & & & \\ \bullet & \bullet & \bullet & & & & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

S thus equals *the area of the height-5, base-9 right triangle.*

Since the base and height of the triangle are both *odd*,
the area is not easy to calculate visually.

(We prefer to avoid any textual calculation.)

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number-oriented* proof: STEP 4

$$S \;=\; \begin{array}{ccccccccc} \bullet & & & & & & & & \\ \bullet & \bullet & \bullet & & & & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & & & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & & \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet \end{array} \;=\; \text{AREA(height-}5\text{, base-}9\text{ right triangle)}$$

To make this area easy to compute visually
We augment the sum-triangle with a "shadow" version:

$$2S \;=\; \begin{array}{cccccccccc} \bullet & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \bullet & \bullet & \bullet & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\ \bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ & \circ & \circ \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ \\ \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \circ \end{array}$$

The area of the right triangle is one-half the area of the rectangle!

—and the base of this rectangle is *even*!

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number-oriented* proof: STEP 4

$$2S \quad = \quad \begin{array}{cccccccccc}
\bullet & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\bullet & \bullet & \bullet & \circ & \circ & \circ & \circ & \circ & \circ & \circ \\
\bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ & \circ & \circ \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \circ
\end{array}$$

The area of the right triangle is one-half the area of the rectangle!

—and the base of this rectangle is *even*!

$$S \quad = \quad \frac{1}{2} \times 5 \times 10 \;=\; \frac{1}{2} \times 50 \;=\; 25 \;=\; 5^2$$

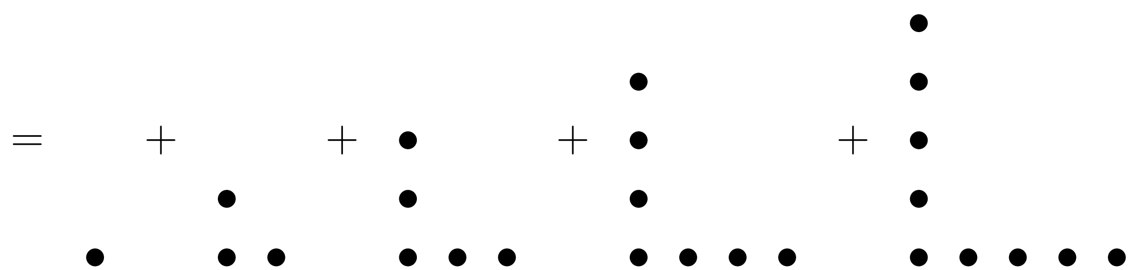
A TOKEN-NUMBER+CONFIGURATION-ORIENTED PROOF

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 1

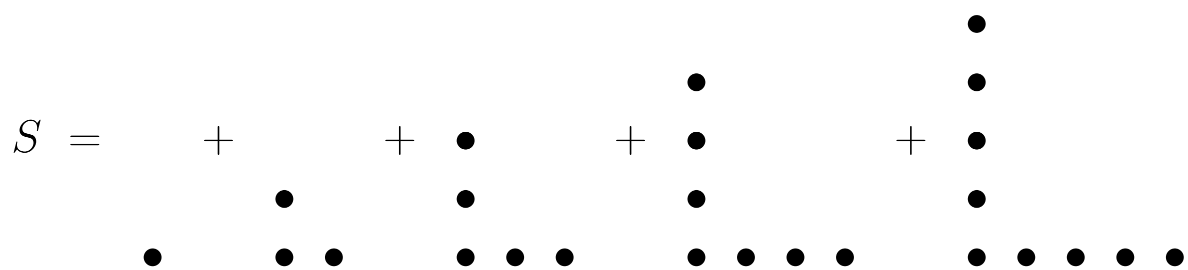
Write out the summaton of interest:

$$S = 1 + 3 + 5 + 7 + 9$$

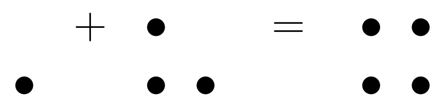


Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 2



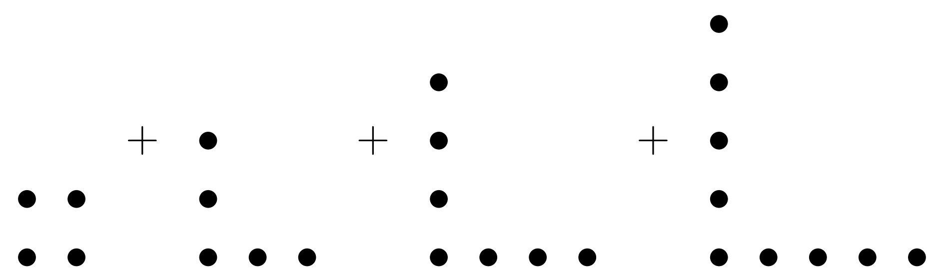
Note that the first two summands sum by *nesting*:



Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 3

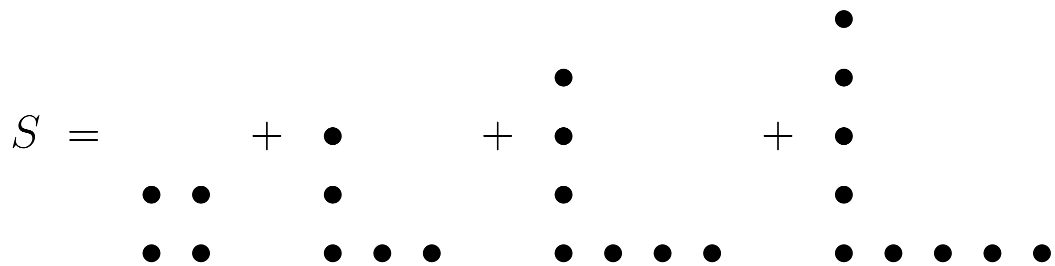
$S =$



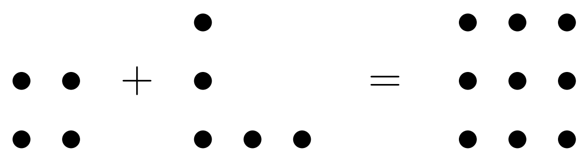
$$= 4 + 5 + 7 + 9$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 3



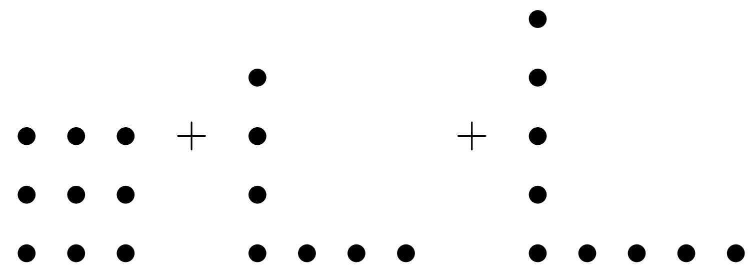
Note that the first two summands sum by *nesting*:



Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 4

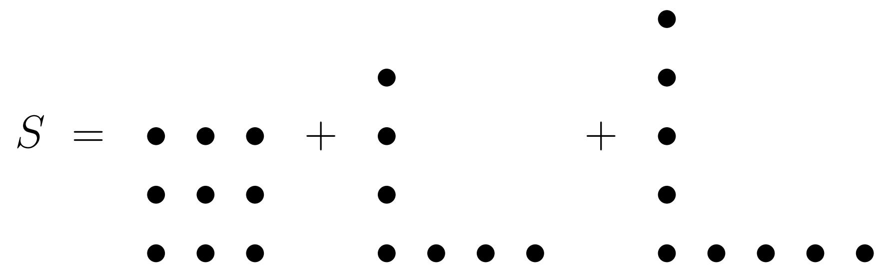
$S =$



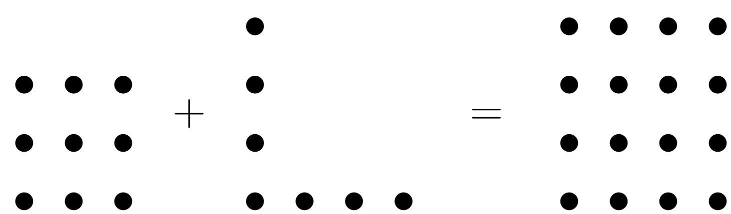
$$= 9 + 7 + 9$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 4



Note that the first two summands sum by *nesting*:



Each perfect square n^2 is the sum of the first n odd numbers.

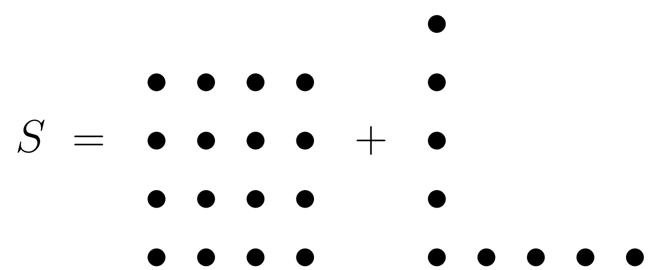
A *token-number+configuration-oriented* proof: STEP 5

$$
\begin{array}{ccccccccccccc}
 & & & & & & & & \bullet & & & & \\
 & & \bullet & \bullet & \bullet & \bullet & & & \bullet & & & & \\
S & = & \bullet & \bullet & \bullet & \bullet & + & & \bullet & & & & \\
 & & \bullet & \bullet & \bullet & \bullet & & & \bullet & & & & \\
 & & \bullet & \bullet & \bullet & \bullet & & & \bullet & \bullet & \bullet & \bullet & \bullet
\end{array}
$$

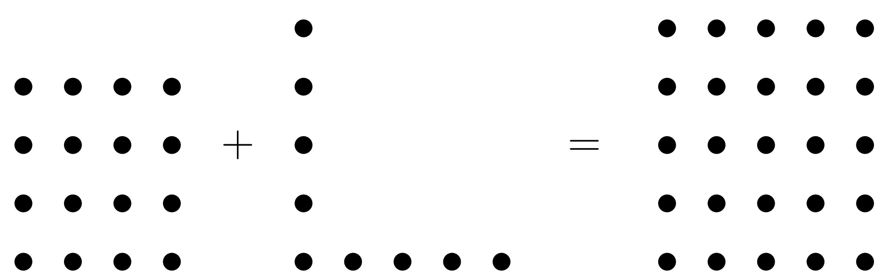
$$= \; 16 \; + \; 9$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 5



Note that the final two summands sum by *nesting*:



Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 6

$$S = \begin{array}{ccccc} \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

$$= 25$$

$$= 5^2$$

Each perfect square n^2 is the sum of the first n odd numbers.

A *token-number+configuration-oriented* proof: STEP 6

$$S \;=\; \begin{array}{ccccc} \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \end{array}$$

$$= \; 25$$

$$= \; 5^2$$

As before, this illustration extends to arbitrary positive integers n

Situation $\#2$

HOW TO THINK ABOUT *COMPUTATIONS*

Situation #2

HOW TO THINK ABOUT *COMPUTATIONS*

The main message:

Step back from details in algorithms and programs.

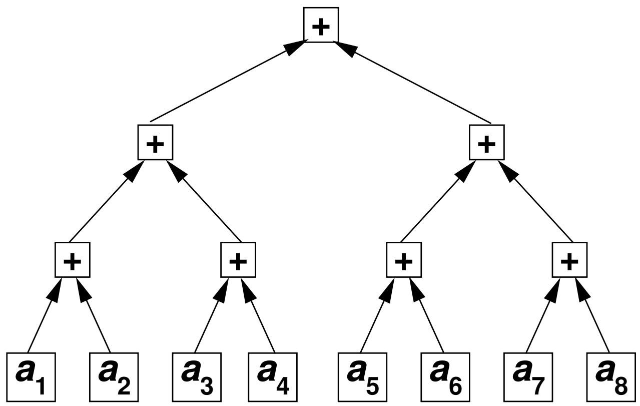
You may observe computationally advantageous nonobvious similarities.

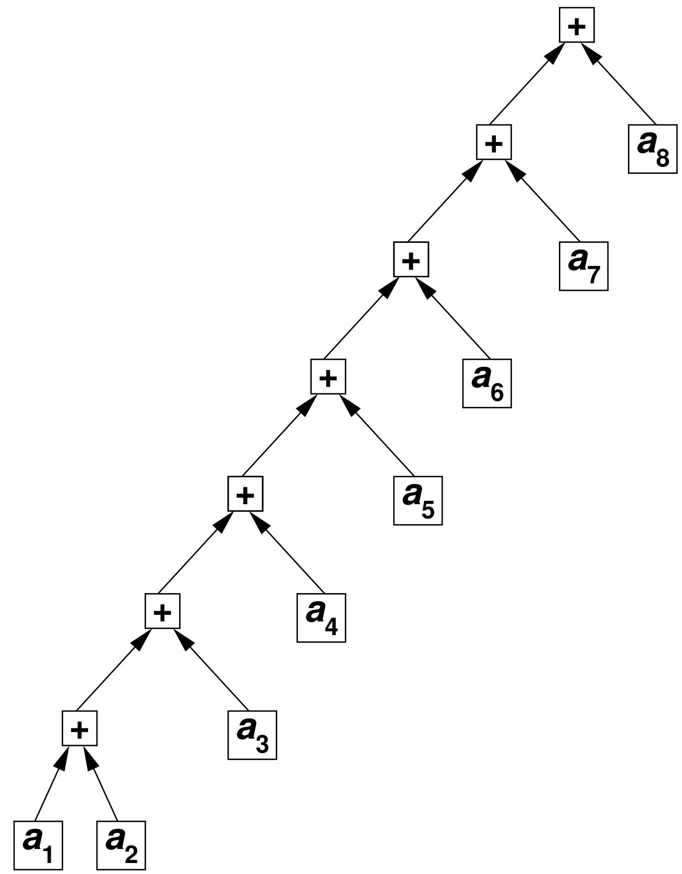
Tasks in a computation cannot be executed in arbitrary orders.

Tasks in a computation cannot be executed in arbitrary orders.

—Some tasks depend on inputs from other tasks

Two ways to sum 8 numbers — illustrating dependencies





Tasks in a computation cannot be executed in arbitrary orders.

—Some tasks depend on inputs from other tasks

Tasks in a computation cannot be executed in arbitrary orders.

—Some tasks depend on inputs from other tasks

Dependencies constrain

- a *compiler*'s freedom to rearrange orders of task-execution

  —to optimize access to/consumption of resources

Tasks in a computation cannot be executed in arbitrary orders.

—Some tasks depend on inputs from other tasks

Dependencies constrain

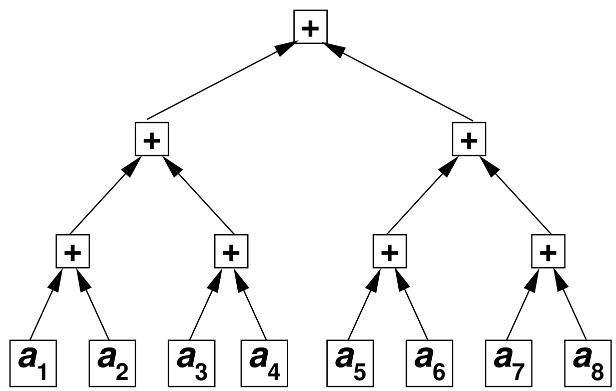
- a $compiler$'s freedom to rearrange orders of task-execution
- a $scheduler$'s freedom to have tasks execute concurrently

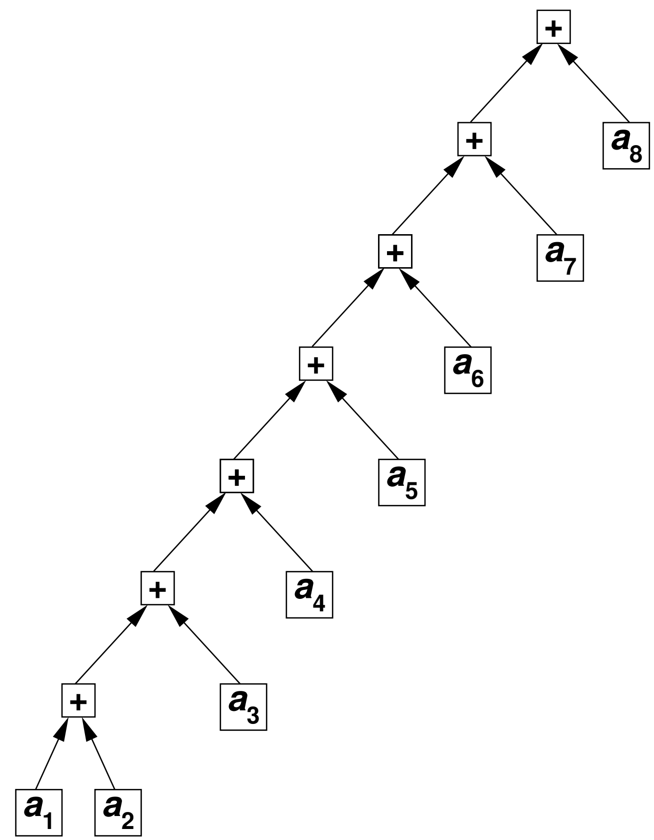
  —to optimize cumulative task-execution time

Dependencies constrain

- a *compiler*'s freedom to rearrange orders of task-execution
- a *scheduler*'s freedom to have tasks execute concurrently

The degree of constraint can vary considerably





Because of its importance to compilation and scheduling ...

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- the design of *efficient* algorithms

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- the design of *efficient* algorithms

  –all kinds of efficiency:

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- the design of *efficient* algorithms

  –all kinds of efficiency:

  ⋆ operation speed

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- the design of *efficient* algorithms

  –all kinds of efficiency:

  - $\star$ operation speed
  - $\star$ utilization of memory/storage

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- the design of *efficient* algorithms

  –all kinds of efficiency:

  - $\star$ operation speed
  - $\star$ utilization of memory/storage
  - $\star$ conservation of power

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- The design of *efficient* algorithms
- the design of high-performance programs

Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

- The design of *efficient* algorithms
- the design of high-performance programs
- understanding that

  many algorithms, for many disparate tasks, <u>share the same dependency structure</u>

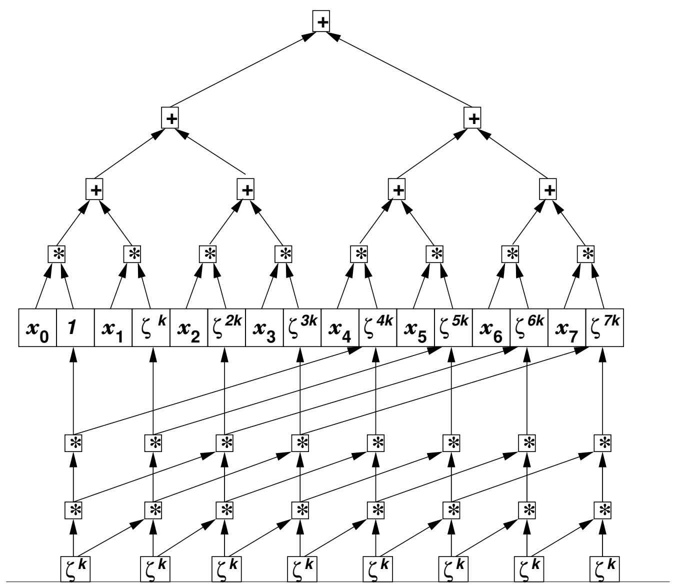
Because of its importance to compilation and scheduling ...

understanding dependency structures is critical to

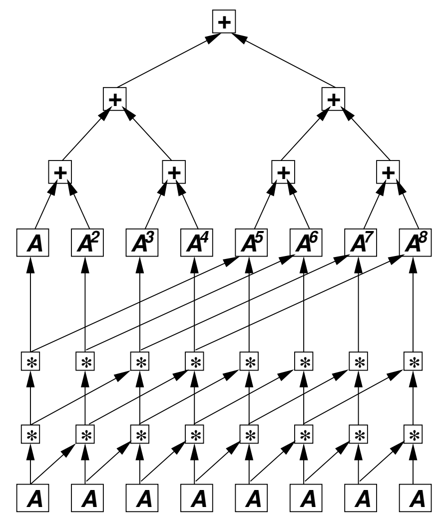
- The design of *efficient* algorithms
- the design of high-performance programs
- understanding that
  many algorithms, for many disparate tasks, share the same dependency structure

Some examples:

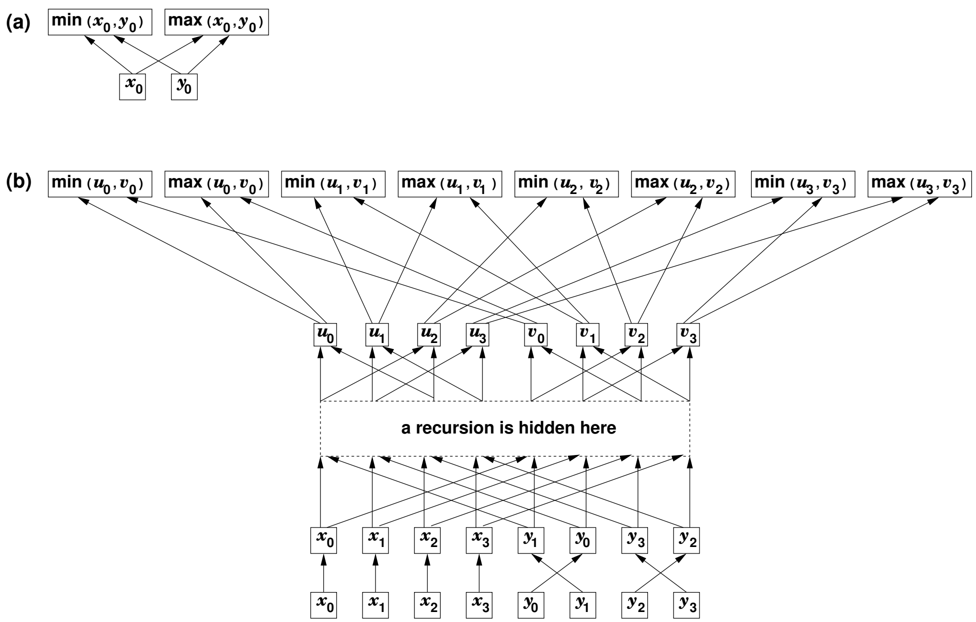
The Discrete Laplace Transform



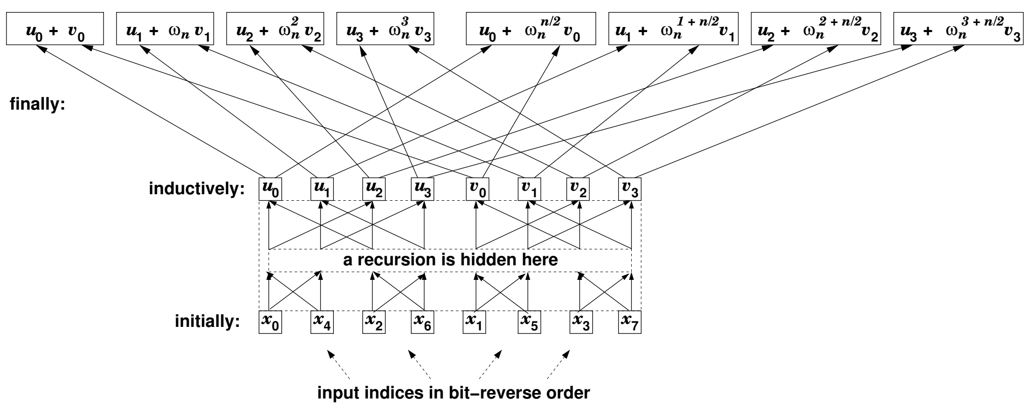
Finding all paths in a graph



A butterfly sorting algorithm



The Fast Fourier Transform



Situation #3

HOW TO THINK ABOUT *COMPUTING*

Situation #3

HOW TO THINK ABOUT *COMPUTING*

The main message:

The integer I have written behind this screen is actually . . .

Situation #3

HOW TO THINK ABOUT *COMPUTING*

The main message:

The integer I have written behind this screen is actually . . .

- just a number?

Situation #3

HOW TO THINK ABOUT ***COMPUTING***

The main message:

The integer I have written behind this screen is actually . . .

- just a number?
- an encoding of a program that computes π to $10^{10^{10}}$ places?

Situation #3

HOW TO THINK ABOUT *COMPUTING*

The main message:

The integer I have written behind this screen is actually . . .

- just a number?
- an encoding of a program that computes π to $10^{10^{10}}$ places?
- an encoding of the computation by that program?

Situation $\#3$

HOW TO THINK ABOUT *COMPUTING*

The main message:

The integer I have written behind this screen is actually ...

- just a number?
- an encoding of a program that computes π to $10^{10^{10}}$ places?
- an encoding of the computation by that program?
- an encoding of the elements of the set of all sets that can be written using $10^{10^{10}}$ characters?

Situation #3

HOW TO THINK ABOUT <u>*COMPUTING*</u>

The main message:

The integer I have written behind this screen is actually . . .

- just a number?
- an encoding of a program that computes π to $10^{10^{10}}$ places?
- an encoding of the computation by that program?
- an encoding of the elements of the set of all sets that can be written using $10^{10^{10}}$ characters?

ENCODINGS ARE POWERFUL — and mischievous — MECHANISMS

that span the chasm between <u>NUMBER</u> and <u>PROGRAM</u> and <u>LANGUAGE</u>

- Kurt Gödel:

  "This sentence is false"

- Kurt Gödel:

  "This sentence is false"

IS THIS SENTENCE TRUE?

- Kurt Gödel:

  "This sentence is false"

IS THIS SENTENCE TRUE?

- If it is true — then it is false!

- Kurt Gödel:

  “This sentence is false”

IS THIS SENTENCE TRUE?

- If it is true — then it is false!
- If it is false — then it is true!

- Kurt Gödel:

  "This sentence is false"

IS THIS SENTENCE TRUE?

- If it is true — then it is false!
- If it is false — then it is true!

This curious ***self-referential*** sentence teaches us:

$$\boxed{\textit{Not all assertions are } \mathrm{TRUE} \textit{ or } \mathrm{FALSE}.}$$

- Kurt Gödel:

  "This sentence is false"

IS THIS SENTENCE TRUE?

- If it is true — then it is false!
- If it is false — then it is true!

This curious *self-referential* sentence teaches us:

Not all assertions are TRUE *or* FALSE.

Not all TRUE *assertions are* PROVABLE.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  Let X be the set whose elements are
  all sets that *are not* elements of themselves

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  Let X be the set whose elements are
  <u>all sets that *are not* elements of themselves</u>

Note that sets *can* have sets as elements!

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  Let X be the set whose elements are
  <u>all sets that *are not* elements of themselves</u>

IS X AN ELEMENT OF X?

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  Let X be the set whose elements are
  <u>all sets that *are not* elements of themselves</u>

IS X AN ELEMENT OF X?

- If X is an element of X, then X is *not* an element of X

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  Let X be the set whose elements are
  all sets that *are not* elements of themselves

IS X AN ELEMENT OF X?

- If X is an element of X, then X is *not* an element of X
- If X is *not* an element of X, then X is an element of X.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  Let X be the set whose elements are
  <u>all sets that *are not* elements of themselves</u>

IS X AN ELEMENT OF X?

- If X is an element of X, then X is *not* an element of X
- If X is *not* an element of X, then X is an element of X.

This $\textit{self-referential}$ paradox teaches us:

$$\boxed{\textit{Set } X \textit{ cannot exist.}}$$

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs ...

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs ...

Clearly such programs exist – a compiler is an example.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs ...
  AND
  that enters an endless loop *precisely* when
  an input program Π *halts* on input Π.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs . . .
  AND
  <u>that enters an endless loop *precisely* when</u>
  <u>an input program Π *halts* on input Π.</u>

DOES PROGRAM $\mathbf{P}$ HALT ON INPUT $\mathbf{P}$?

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs ...
  AND
  <u>that enters an endless loop *precisely* when</u>
  <u>an input program Π *halts* on input Π.</u>

DOES PROGRAM $\mathbf{P}$ HALT ON INPUT $\mathbf{P}$?

- If program $\mathbf{P}$ halts on input $\mathbf{P}$, then it *does not* halt on input $\mathbf{P}$.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs ...
  AND
  that enters an endless loop *precisely* when
  an input program Π *halts* on input Π.

DOES PROGRAM $\mathbf{P}$ HALT ON INPUT $\mathbf{P}$?

- If program $\mathbf{P}$ halts on input $\mathbf{P}$, then it *does not* halt on input $\mathbf{P}$.

- If program $\mathbf{P}$ does not halt on input $\mathbf{P}$, then it *does* halt on input $\mathbf{P}$.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  Let $\mathbf{P}$ be a program that takes programs as inputs ...
  AND
  <u>that enters an endless loop *precisely* when</u>
  <u>an input program Π *halts* on input Π.</u>

DOES PROGRAM $\mathbf{P}$ HALT ON INPUT $\mathbf{P}$?

- If program $\mathbf{P}$ halts on input $\mathbf{P}$, then it *does not* halt on input $\mathbf{P}$.

- If program $\mathbf{P}$ does not halt on input $\mathbf{P}$, then it *does* halt on input $\mathbf{P}$.

This *self-referential* paradox teaches us:

$\boxed{\textit{Program } \mathbf{P} \textit{ cannot exist.}}$

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  one-to-one correspondence between the integers and the real numbers

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  <u>one-to-one correspondence between the integers and the real numbers</u>

In detail, I mean that the sequence

$$\mathbf{P}(1),\ \mathbf{P}(2),\ \mathbf{P}(3),\ \ldots$$

contains every real number *precisely once*.

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  <u>one-to-one correspondence between the integers and the real numbers</u>

I can write a program Π *such that* $\Pi(\mathbf{P})$

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  <u>one-to-one correspondence between the integers and the real numbers</u>

I can write a program Π *such that* $\Pi(\mathbf{P})$

–i.e., the result produced by program Π on input $\mathbf{P}$

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  <u>one-to-one correspondence between the integers and the real numbers</u>

I can write a program Π *such that* $\Pi(\mathbf{P})$

is a real number that is <u>not on your list!</u>

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  <u>one-to-one correspondence between the integers and the real numbers</u>

I can write a program Π *such that* $\Pi(\mathbf{P})$

is a real number that is <u>not on your list!</u>

The lesson is:

$\boxed{\textit{There is no one-to-one correspondence between the integers and the real numbers}}$

- Kurt Gödel:

  "This sentence is false"

- Bertrand Russell:

  X *is the set of all sets that* are not *elements of themselves*

- Alan Turing:

  $\mathbf{P}(\Pi)$ *fails to halt* just when $\Pi(\Pi)$ *halts*.

- Georg Cantor:

  Say that you present me with a program $\mathbf{P}$ that produces a
  one-to-one correspondence between the integers and the real numbers

I can write a program Π *such that* $\Pi(\mathbf{P})$

is a real number that is not on your list!

The lesson is:

$\boxed{\textit{There is no one-to-one correspondence between the integers and the real numbers}}$

≈≈≈≈≈ AND ≈≈≈≈≈

≈≈≈≈≈ AND ≈≈≈≈≈

$\boxed{\textit{ALL OF THE PROOFS}}$

— Gödel's, Russells', Turing's, and Cantor's —

$\boxed{\textit{ARE ENCODINGS OF ONE ANOTHER!}}$