3. Mathematical Induction & Recursive Definitions

3. Mathematical Induction & Recursive Definitions

a. Intro to Math. Induction

A Super-natural example!

- You're doing time in purgatory, and are automatically enrolled in philosophy (lucky you!)

A Super-natural example!

- You're doing time in purgatory, and are automatically enrolled in philosophy (lucky you!)
- Good news: To pass your class, you just need to produce one satisfactory piece of work

A Super-natural example!

- You’re doing time in purgatory, and are automatically enrolled in philosophy (lucky you!)
- Good news: To pass your class, you just need to produce one satisfactory piece of work
- Bad news bears: your teacher is Gordon Ramsay, and he is never satisfied with any assignment

A Super-natural example!

- You're doing time in purgatory, and are automatically enrolled in philosophy (lucky you!)
- Good news: To pass your class, you just need to produce one satisfactory piece of work
- Bad news bears: your teacher is Gordon Ramsay, and he is never satisfied with any assignment
- Whenever you turn something in, Ramsay returns it with suggestions for improvement, with revisions due tomorrow

A Super-natural example!

- You're doing time in purgatory, and are automatically enrolled in philosophy (lucky you!)
- Good news: To pass your class, you just need to produce one satisfactory piece of work
- Bad news bears: your teacher is Gordon Ramsay, and he is never satisfied with any assignment
- Whenever you turn something in, Ramsay returns it with suggestions for improvement, with revisions due tomorrow
- How long should you expect to spend in this class?

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property ('**base case**')

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property ('**base case**')
2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
   '**Induction step**' (There are five cases to consider:)

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property ('**base case**')
2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
   '**Induction step**' (There are five cases to consider:)
   (i) $\sim\mathscr{A}$

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property (‘**base case**’)
2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective

   ‘**Induction step**’ (There are five cases to consider:)

   (i) $\sim\mathscr{A}$

   (ii) $(\mathscr{A} \mathbin{\&} \mathscr{B})$

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property (‘**base case**’)
2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective

   ‘**Induction step**’ (There are five cases to consider:)

   (i) $\sim\mathscr{A}$

   (ii) $(\mathscr{A} \,\&\, \mathscr{B})$

   (iii) $(\mathscr{A} \vee \mathscr{B})$

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property (‘**base case**’)
2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
   ‘**Induction step**’ (There are five cases to consider:)
   (i) $\sim\mathscr{A}$
   (ii) $(\mathscr{A} \,\&\, \mathscr{B})$
   (iii) $(\mathscr{A} \vee \mathscr{B})$
   (iv) $(\mathscr{A} \supset \mathscr{B})$

Mathematical Induction for SL Sentences

If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

1. All **atomic** sentences have that property ('**base case**')
2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
   '**Induction step**' (There are five cases to consider:)
   (i) $\sim\mathscr{A}$
   (ii) $(\mathscr{A} \,\&\, \mathscr{B})$
   (iii) $(\mathscr{A} \vee \mathscr{B})$
   (iv) $(\mathscr{A} \supset \mathscr{B})$
   (v) $(\mathscr{A} \equiv \mathscr{B})$

A Simple Example

- Prove by induction that every well-formed formulae (wff) of SL has exactly as many left parentheses as it has right parentheses

- Don't forget to explicitly state the **base case**

- Don't forget to explicitly state the **Induction Step**!

Three More Examples!

Prove the following by induction. Don't forget to explicitly state the base case and the induction step!

1. No wff begins or ends with a binary connective

Three More Examples!

Prove the following by induction. Don’t forget to explicitly state the base case and the induction step!

1. No wff begins or ends with a binary connective
2. No wff contains two consecutive binary connectives (i.e. with no symbols between them)

Three More Examples!

Prove the following by induction. Don’t forget to explicitly state the base case and the induction step!

1. No wff begins or ends with a binary connective
2. No wff contains two consecutive binary connectives (i.e. with no symbols between them)
3. If a wff doesn’t contain any binary connectives, then it is contingent. (hint: say that a wff is *baller* if it either contains a binary connective or is contingent. Use induction to show that every wff is baller.)

Two DISTINCT notions of ‘Induction’

- **Mathematical Induction** is completely different from **scientific induction** or what we earlier called ‘inductive arguments’ (which we contrasted with ‘deductive arguments’)

Two DISTINCT notions of ‘Induction’

- **Mathematical Induction** is completely different from **scientific induction** or what we earlier called ‘inductive arguments’ (which we contrasted with ‘deductive arguments’)
- **Mathematical Induction**: Rigorous method of proving that a property holds for an infinite number of cases. Follow the steps we’ve outlined!

Two DISTINCT notions of ‘Induction’

- **Mathematical Induction** is completely different from **scientific induction** or what we earlier called ‘inductive arguments’ (which we contrasted with ‘deductive arguments’)
- **Mathematical Induction**: Rigorous method of proving that a property holds for an infinite number of cases. Follow the steps we’ve outlined!
- **Scientific Induction**: fallible method of supporting a claim, based on a finite number of previous observations.

Two DISTINCT notions of ‘Induction’

- **Mathematical Induction** is completely different from **scientific induction** or what we earlier called ‘inductive arguments’ (which we contrasted with ‘deductive arguments’)
- **Mathematical Induction**: Rigorous method of proving that a property holds for an infinite number of cases. Follow the steps we’ve outlined!
- **Scientific Induction**: fallible method of supporting a claim, based on a finite number of previous observations.
  - All of the ravens I have ever seen are black

Two DISTINCT notions of 'Induction'

- **Mathematical Induction** is completely different from **scientific induction** or what we earlier called 'inductive arguments' (which we contrasted with 'deductive arguments')
- **Mathematical Induction**: Rigorous method of proving that a property holds for an infinite number of cases. Follow the steps we've outlined!
- **Scientific Induction**: fallible method of supporting a claim, based on a finite number of previous observations.
    - All of the ravens I have ever seen are black
    - Therefore, the next raven I see will be black

Two DISTINCT notions of ‘Induction’

- **Mathematical Induction** is completely different from **scientific induction** or what we earlier called ‘inductive arguments’ (which we contrasted with ‘deductive arguments’)
- **Mathematical Induction**: Rigorous method of proving that a property holds for an infinite number of cases. Follow the steps we’ve outlined!
- **Scientific Induction**: fallible method of supporting a claim, based on a finite number of previous observations.
  - All of the ravens I have ever seen are black
  - Therefore, the next raven I see will be black
  - Notice how your conclusion could be wrong! Albino ravens!

3. Mathematical Induction & Recursive Definitions

b. Recursive Definitions

A Non-recursive Motivation for Recursion

- When a bigger thing can be studied in smaller chunks, it is often fruitful to do so

A Non-recursive Motivation for Recursion

- When a bigger thing can be studied in smaller chunks, it is often fruitful to do so
- Descartes' *Rules for the Direction of the Mind*:
  break problems down to their simplest parts and have a secure grasp of them before proceeding

A Non-recursive Motivation for Recursion

- When a bigger thing can be studied in smaller chunks, it is often fruitful to do so
- Descartes' *Rules for the Direction of the Mind*:
  break problems down to their simplest parts and have a secure grasp of them before proceeding
- Or as we might say: Show me complexity, and I will find simplicity (Gotta get this on a T-Shirt!)

A Non-recursive Motivation for Recursion

- When a bigger thing can be studied in smaller chunks, it is often fruitful to do so
- Descartes' *Rules for the Direction of the Mind*:
  break problems down to their simplest parts and have a secure grasp of them before proceeding
- Or as we might say: Show me complexity, and I will find simplicity (Gotta get this on a T-Shirt!)
- Recursive definitions are one instance of applying this rule

A Non-recursive approach to Language Learning

- Some tourist manuals will have lists of sentences to phonetically memorize before travelling: learn how to say "Where is the train to the airport?" or "Where can one find a drugstore?" etc.

A Non-recursive approach to Language Learning

- Some tourist manuals will have lists of sentences to phonetically memorize before travelling: learn how to say "Where is the train to the airport?" or "Where can one find a drugstore?" etc.
- The plan is to commit all these to memory, by brute force.

A Non-recursive approach to Language Learning

- Some tourist manuals will have lists of sentences to phonetically memorize before travelling: learn how to say “Where is the train to the airport?” or “Where can one find a drugstore?” etc.
- The plan is to commit all these to memory, by brute force.
- You will know what “Hvor er toget til lufthavnen”, “Hvor finder man et apotek?” etc. mean because you have seen them before and can recall them.

A Non-recursive approach to Language Learning

- Some tourist manuals will have lists of sentences to phonetically memorize before travelling: learn how to say "Where is the train to the airport?" or "Where can one find a drugstore?" etc.
- The plan is to commit all these to memory, by brute force.
- You will know what "Hvor er toget til lufthavnen", "Hvor finder man et apotek?" etc. mean because you have seen them before and can recall them.
- But this is entirely uncreative. You can only say/understand things you have seen before using this technique.

Recursion as the basis of English

- But much of the time, we say things that we've *never* said or heard before, yet we understand what they mean.

Recursion as the basis of English

- But much of the time, we say things that we've *never* said or heard before, yet we understand what they mean.
  - "Your fate awaits in the office of the chair of the Corporation"

Recursion as the basis of English

- But much of the time, we say things that we've *never* said or heard before, yet we understand what they mean.
  - "Your fate awaits in the office of the chair of the Corporation"
  - I would bet no one in class has heard this sentence before.

Recursion as the basis of English

- But much of the time, we say things that we've *never* said or heard before, yet we understand what they mean.
  - "Your fate awaits in the office of the chair of the Corporation"
  - I would bet no one in class has heard this sentence before.
  - But (maybe) you understand what it means.

Recursion as the basis of English

- But much of the time, we say things that we've *never* said or heard before, yet we understand what they mean.
  - "Your fate awaits in the office of the chair of the Corporation"
  - I would bet no one in class has heard this sentence before.
  - But (maybe) you understand what it means.
- What makes this possible is that our language has a recursive structure – we have basic components and then rules for making more complex, meaningful expressions out of them.

Example: Possessive of a noun phrase

- We can iterate grammatical rules, i.e. apply them again and again.

Example: Possessive of a noun phrase

- We can iterate grammatical rules, i.e. apply them again and again.
- For example we have a way to form an expression "[Noun Phrase]'s son".

Example: Possessive of a noun phrase

- We can iterate grammatical rules, i.e. apply them again and again.
- For example we have a way to form an expression “[Noun Phrase]’s son”.
- I can apply this to the name ‘Andrea’ to get “Andrea’s son”.

Example: Possessive of a noun phrase

- We can iterate grammatical rules, i.e. apply them again and again.
- For example we have a way to form an expression "[Noun Phrase]'s son".
- I can apply this to the name 'Andrea' to get "Andrea's son".
- Then I can apply it again, to get "Andrea's son's son".

Example: Possessive of a noun phrase

- We can iterate grammatical rules, i.e. apply them again and again.
- For example we have a way to form an expression “[Noun Phrase]’s son”.
- I can apply this to the name ‘Andrea’ to get “Andrea’s son”.
- Then I can apply it again, to get “Andrea’s son’s son”.
- And again, to get “Andrea’s son’s son’s son”.

Example: nested object of verb phrases

- Sometimes, sentences can be so complicated that we need to break them down into components and study their structure to figure out what they mean

Example: nested object of verb phrases

- Sometimes, sentences can be so complicated that we need to break them down into components and study their structure to figure out what they mean
- But if we understand the words, and we understand the rules of grammar, we can figure out the recursive structure

Example: nested object of verb phrases

- Sometimes, sentences can be so complicated that we need to break them down into components and study their structure to figure out what they mean
- But if we understand the words, and we understand the rules of grammar, we can figure out the recursive structure
- For example, the following is a meaningful sentence of English:

Example: nested object of verb phrases

- Sometimes, sentences can be so complicated that we need to break them down into components and study their structure to figure out what they mean
- But if we understand the words, and we understand the rules of grammar, we can figure out the recursive structure
- For example, the following is a meaningful sentence of English:
- *The rat the cat the dog bit chased ate the cheese*

Example: nested object of verb phrases

- Sometimes, sentences can be so complicated that we need to break them down into components and study their structure to figure out what they mean
- But if we understand the words, and we understand the rules of grammar, we can figure out the recursive structure
- For example, the following is a meaningful sentence of English:
- *The rat the cat the dog bit chased ate the cheese*
- What is this rat smoking??? It’s not obvious that this is a meaningful sentence, never mind what it means!

Example: nested object of verb phrases

- Sometimes, sentences can be so complicated that we need to break them down into components and study their structure to figure out what they mean
- But if we understand the words, and we understand the rules of grammar, we can figure out the recursive structure
- For example, the following is a meaningful sentence of English:
- *The rat the cat the dog bit chased ate the cheese*
- What is this rat smoking??? It's not obvious that this is a meaningful sentence, never mind what it means!

Break it down!

- To figure out what it's saying, let's take a simpler part. Say that you see a rat eating some cheese.

Break it down!

- To figure out what it's saying, let's take a simpler part. Say that you see a rat eating some cheese.
- You could say: "The rat ate the cheese." Nothing obscure here

Break it down!

- To figure out what it's saying, let's take a simpler part. Say that you see a rat eating some cheese.
- You could say: "The rat ate the cheese." Nothing obscure here
- But maybe there were several rats running around, and only one of them ate the cheese.

Break it down!

- To figure out what it's saying, let's take a simpler part. Say that you see a rat eating some cheese.
- You could say: "The rat ate the cheese." Nothing obscure here
- But maybe there were several rats running around, and only one of them ate the cheese.
- You want to indicate which one, and it happens to be the one that the cat chased.

Break it down!

- To figure out what it’s saying, let’s take a simpler part. Say that you see a rat eating some cheese.
- You could say: “The rat ate the cheese.” Nothing obscure here
- But maybe there were several rats running around, and only one of them ate the cheese.
- You want to indicate which one, and it happens to be the one that the cat chased.
- So you can form the noun phrase “The rat the cat chased”, and state “The rat the cat chased ate the cheese.”

Break it down!

- To figure out what it's saying, let's take a simpler part. Say that you see a rat eating some cheese.
- You could say: "The rat ate the cheese." Nothing obscure here
- But maybe there were several rats running around, and only one of them ate the cheese.
- You want to indicate which one, and it happens to be the one that the cat chased.
- So you can form the noun phrase "The rat the cat chased", and state "The rat the cat chased ate the cheese."
- The rat $\underbrace{\Longrightarrow\Longrightarrow}_{\text{Which rat?}}$ The rat the cat chased

Still breaking...

- And perhaps there are lots of cats running around too.

Still breaking...

- And perhaps there are lots of cats running around too.
- You want to say specifically that you are referring to the cat that the dog bit.

  The cat $\underbrace{\Longrightarrow\Longrightarrow}_{\text{Which cat?}}$ The cat the dog bit.

  So take the phrase “The rat the cat chased”, and replace “the cat” with “the cat the dog bit” to get the noun phrase:

Still breaking...

- And perhaps there are lots of cats running around too.
- You want to say specifically that you are referring to the cat that the dog bit.

  The cat $\underbrace{\Longrightarrow\Longrightarrow}_{\text{Which cat?}}$ The cat the dog bit.

  So take the phrase "The rat the cat chased", and replace "the cat" with "the cat the dog bit" to get the noun phrase:
- "The rat the cat the dog bit chased":

Still breaking...

- And perhaps there are lots of cats running around too.
- You want to say specifically that you are referring to the cat that the dog bit.

  The cat $\underbrace{\Longrightarrow\Longrightarrow}_{\text{Which cat?}}$ The cat the dog bit.

  So take the phrase "The rat the cat chased", and replace "the cat" with "the cat the dog bit" to get the noun phrase:
- "The rat the cat the dog bit chased":
- And you can state that this particular rat, chased by that particular cat [the cat the dog bit] ate the cheese:

Still breaking...

- And perhaps there are lots of cats running around too.
- You want to say specifically that you are referring to the cat that the dog bit.

  The cat $\underbrace{\Longrightarrow\Longrightarrow}_{\text{Which cat?}}$ The cat the dog bit.

  So take the phrase “The rat the cat chased”, and replace “the cat” with “the cat the dog bit” to get the noun phrase:
- “The rat the cat the dog bit chased”:
- And you can state that this particular rat, chased by that particular cat [the cat the dog bit] ate the cheese:
- “The rat the cat the dog bit chased ate the cheese.”

Still breaking...

- And perhaps there are lots of cats running around too.
- You want to say specifically that you are referring to the cat that the dog bit.

  The cat $\underbrace{\Longrightarrow\Longrightarrow}_{\text{Which cat?}}$ The cat the dog bit.

  So take the phrase “The rat the cat chased”, and replace “the cat” with “the cat the dog bit” to get the noun phrase:
- “The rat the cat the dog bit chased”:
- And you can state that this particular rat, chased by that particular cat [the cat the dog bit] ate the cheese:
- “The rat the cat the dog bit chased ate the cheese.”

More Examples of our Beautiful Recursive Language

- Now that you know the rules, and can apply them over and over, you can figure out what this means.

More Examples of our Beautiful Recursive Language

- Now that you know the rules, and can apply them over and over, you can figure out what this means.
- "The rat the cat the dog the farmer bought bit chased ate the cheese."

More Examples of our Beautiful Recursive Language

- Now that you know the rules, and can apply them over and over, you can figure out what this means.
- "The rat the cat the dog the farmer bought bit chased ate the cheese."
- Some others to try for yourself (not essential to the course, for entertainment value only:)

More Examples of our Beautiful Recursive Language

- Now that you know the rules, and can apply them over and over, you can figure out what this means.
- "The rat the cat the dog the farmer bought bit chased ate the cheese."
- Some others to try for yourself (not essential to the course, for entertainment value only:)
- Fish fish fish fish fish.
  Fish fish fish fish fish fish fish

A Moral of our Convoluted Story

- **The point**: if you know the basic units (in this case: the words; in *SL* the sentence letters) and you know the rules for creating more complex meaningful parts, you open up an astonishing range of possibilities.

A Moral of our Convoluted Story

- **The point**: if you know the basic units (in this case: the words; in *SL* the sentence letters) and you know the rules for creating more complex meaningful parts, you open up an astonishing range of possibilities.
- Definition by Recursion is the canonical way to define objects or structures that depend on iterated rules applied to a given basis

A Moral of our Convoluted Story

- **The point**: if you know the basic units (in this case: the words; in *SL* the sentence letters) and you know the rules for creating more complex meaningful parts, you open up an astonishing range of possibilities.
- Definition by Recursion is the canonical way to define objects or structures that depend on iterated rules applied to a given basis
- Proof by Induction is a powerful way to prove things about structures that are defined in this way.

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
    - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
    - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$
    - If $\mathscr{P}$ and $\mathbb{Q}$ are both wffs, then so are:

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
    - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$
    - If $\mathscr{P}$ and $\mathbb{Q}$ are both wffs, then so are:
        - $(\mathscr{P} \,\&\, \mathbb{Q})$

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
    - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$
    - If $\mathscr{P}$ and $\mathscr{Q}$ are both wffs, then so are:
        - $(\mathscr{P} \,\&\, \mathscr{Q})$
        - $(\mathscr{P} \vee \mathscr{Q})$

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
   - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$
   - If $\mathscr{P}$ and $\mathbb{Q}$ are both wffs, then so are:
     - $(\mathscr{P} \& \mathbb{Q})$
     - $(\mathscr{P} \vee \mathbb{Q})$
     - $(\mathscr{P} \supset \mathbb{Q})$

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
    - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$
    - If $\mathscr{P}$ and $\mathbb{Q}$ are both wffs, then so are:
        - $(\mathscr{P} \,\&\, \mathbb{Q})$
        - $(\mathscr{P} \vee \mathbb{Q})$
        - $(\mathscr{P} \supset \mathbb{Q})$
        - $(\mathscr{P} \equiv \mathbb{Q})$

Recursive Definition of Sentences of SL

Recall that we define the well-formed formulae (wffs) *recursively*:

1. **Base Clause**: Each atomic sentence is a wff
2. **Recursion Clause(s)**:
   - If $\mathscr{P}$ is a wff, then so is $\sim\mathscr{P}$
   - If $\mathscr{P}$ and $\mathbb{Q}$ are both wffs, then so are:
     - $(\mathscr{P} \,\&\, \mathbb{Q})$
     - $(\mathscr{P} \vee \mathbb{Q})$
     - $(\mathscr{P} \supset \mathbb{Q})$
     - $(\mathscr{P} \equiv \mathbb{Q})$
3. **Closure clause**: Nothing else is a wff of SL

Natural Numbers

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, ...\}$, beginning with 0, is the paradigm of a set of objects with a recursive definition.

Natural Numbers

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, ...\}$, beginning with 0, is the paradigm of a set of objects with a recursive definition.
- The first natural number is 0, and larger numbers are generated by adding 1.

Natural Numbers

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$, beginning with 0, is the paradigm of a set of objects with a recursive definition.
- The first natural number is 0, and larger numbers are generated by adding 1.
- Any whole number greater than 0 can be reached by starting with 0 and iterating this operation.

Natural Numbers

- The set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, ...\}$, beginning with 0, is the paradigm of a set of objects with a recursive definition.
- The first natural number is 0, and larger numbers are generated by adding 1.
- Any whole number greater than 0 can be reached by starting with 0 and iterating this operation.
- (N.B.: Sometimes $\mathbb{N}$ is defined excluding '0';
  but remember: we are inclusive!!!)

Recursive Definition of "natural number"

- Defining the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$:

Recursive Definition of “natural number”

- Defining the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, ...\}$:
- Put the definition into our fave recursion template:

Recursive Definition of “natural number”

- Defining the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$:
- Put the definition into our fave recursion template:
    - **Base clause**: 0 is a natural number.

Recursive Definition of “natural number”

- Defining the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$:
- Put the definition into our fave recursion template:
  - **Base clause**: 0 is a natural number.
  - **Recursion clause**: If n is a natural number then $n + 1$ is a natural number.

Recursive Definition of "natural number"

- Defining the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$:
- Put the definition into our fave recursion template:
  - **Base clause**: 0 is a natural number.
  - **Recursion clause**: If n is a natural number then $n + 1$ is a natural number.
  - **Closure clause**: Nothing else is a natural number.

Recursive Definition of "natural number"

- Defining the set of natural numbers $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$:
- Put the definition into our fave recursion template:
  - **Base clause**: 0 is a natural number.
  - **Recursion clause**: If n is a natural number then $n + 1$ is a natural number.
  - **Closure clause**: Nothing else is a natural number.
- This definition generates the entire set $\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$, starting with 0 and repeating the operation of "$+1$".

Strings! (of letters—not of physics!)

- It can be useful to treat the set of *all strings of letters* in a given alphabet recursively

Strings! (of letters—not of physics!)

- It can be useful to treat the set of *all strings of letters* in a given alphabet recursively
- Say we take a fixed alphabet with just the characters $\{a, b\}$.

Strings! (of letters—not of physics!)

- It can be useful to treat the set of *all strings of letters* in a given alphabet recursively
- Say we take a fixed alphabet with just the characters $\{a, b\}$.
- The basic strings will be the single characters a and b.

Strings! (of letters—not of physics!)

- It can be useful to treat the set of *all strings of letters* in a given alphabet recursively
- Say we take a fixed alphabet with just the characters $\{a, b\}$.
- The basic strings will be the single characters a and b.
- More complex objects are built up by the operation that inputs a string x and a member u of the alphabet, and that outputs the string 'xu' that results from appending 'u' to the right of 'x'

Strings! (of letters—not of physics!)

- It can be useful to treat the set of *all strings of letters* in a given alphabet recursively
- Say we take a fixed alphabet with just the characters $\{a, b\}$.
- The basic strings will be the single characters a and b.
- More complex objects are built up by the operation that inputs a string x and a member u of the alphabet, and that outputs the string 'xu' that results from appending 'u' to the right of 'x'
- Any string can be built up from a single letter by iterating this operation—in effect, simply spelling out 'x' left-to-right.

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:
  - Start with 'b'. Append 'b' to 'b' and obtain 'bb'

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:
  - Start with 'b'. Append 'b' to 'b' and obtain 'bb'
  - Finally, append 'a' to 'bb' and obtain 'bba'

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:
  - Start with 'b'. Append 'b' to 'b' and obtain 'bb'
  - Finally, append 'a' to 'bb' and obtain 'bba'
- Since we can generate every string this way, we can give a recursive definition of "string of letters from the alphabet $\{a, b\}$":

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:
  - Start with 'b'. Append 'b' to 'b' and obtain 'bb'
  - Finally, append 'a' to 'bb' and obtain 'bba'
- Since we can generate every string this way, we can give a recursive definition of "string of letters from the alphabet $\{a, b\}$":
  1. **Base clause**: a and b are strings from $\{a, b\}$

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:
  - Start with 'b'. Append 'b' to 'b' and obtain 'bb'
  - Finally, append 'a' to 'bb' and obtain 'bba'
- Since we can generate every string this way, we can give a recursive definition of "string of letters from the alphabet $\{a, b\}$":
  1. **Base clause**: a and b are strings from $\{a, b\}$
  2. **Recursion clause**: If x is a string from $\{a, b\}$ then xa and xb are strings from $\{a, b\}$

An Example: Recursive Definition of Strings

- Let the alphabet be $\{a, b\}$ and consider the string 'bba': we obtain this string as follows:
  - Start with 'b'. Append 'b' to 'b' and obtain 'bb'
  - Finally, append 'a' to 'bb' and obtain 'bba'
- Since we can generate every string this way, we can give a recursive definition of "string of letters from the alphabet $\{a, b\}$":
  1. **Base clause**: a and b are strings from $\{a, b\}$
  2. **Recursion clause**: If x is a string from $\{a, b\}$ then xa and xb are strings from $\{a, b\}$
  3. **Closure clause**: Nothing else is a string from $\{a, b\}$

Concatenation of the Feline Nation

- A note on terminology: when you stick together two strings of symbols, we say that you “concatenate” them

Concatenation of the Feline Nation

- A note on terminology: when you stick together two strings of symbols, we say that you “concatenate” them
- Sometimes, the symbol ‘$*$’ is used to indicate concatenation, so “$x * y$” means “the result of concatenating the string x and the string y”.

Concatenation of the Feline Nation

- A note on terminology: when you stick together two strings of symbols, we say that you "concatenate" them

- Sometimes, the symbol '$*$' is used to indicate concatenation, so "$x * y$" means "the result of concatenating the string x and the string y".

- Example: '$aaba * cca$' is just the string 'aabacca'

Recursively Defining 'Proofs'

- When we discuss metalogic, we will use the fact that *proofs* provide yet another domain that can be given a recursive definition

Recursively Defining ‘Proofs’

- When we discuss metalogic, we will use the fact that *proofs* provide yet another domain that can be given a recursive definition
- So the set of proofs is another domain where inductive arguments can be given

Recursively Defining ‘Proofs’

- When we discuss metalogic, we will use the fact that *proofs* provide yet another domain that can be given a recursive definition
- So the set of proofs is another domain where inductive arguments can be given
- A generic description for now:

Recursively Defining ‘Proofs’

- When we discuss metalogic, we will use the fact that *proofs* provide yet another domain that can be given a recursive definition
- So the set of proofs is another domain where inductive arguments can be given
- A generic description for now:
- Proofs are constructed by applying rules of inference to premises

Recursively Defining 'Proofs'

- When we discuss metalogic, we will use the fact that *proofs* provide yet another domain that can be given a recursive definition
- So the set of proofs is another domain where inductive arguments can be given
- A generic description for now:
- Proofs are constructed by applying rules of inference to premises
- The simplest proofs are axioms or single premises:

Recursively Defining ‘Proofs’

- When we discuss metalogic, we will use the fact that *proofs* provide yet another domain that can be given a recursive definition
- So the set of proofs is another domain where inductive arguments can be given
- A generic description for now:
- Proofs are constructed by applying rules of inference to premises
- The simplest proofs are axioms or single premises:
  - Any axiom is itself a one-step proof of itself, a single premise is a one-step proof of itself, from itself.

Recursively Defining 'Proofs'

- More complex proofs are built up from simpler ones by adding a step, which must either be an axiom or premise, or follow by a rule from earlier lines in the proof

Recursively Defining ‘Proofs’

- More complex proofs are built up from simpler ones by adding a step, which must either be an axiom or premise, or follow by a rule from earlier lines in the proof
- That is: something is a proof if it is an axiom or premise, or it is obtained from a proof by applying one of a finite number of rules!

Recursively Defining ‘Proofs’

- More complex proofs are built up from simpler ones by adding a step, which must either be an axiom or premise, or follow by a rule from earlier lines in the proof
- That is: something is a proof if it is an axiom or premise, or it is obtained from a proof by applying one of a finite number of rules!
- So it’s a recursive structure: A basis, iterated construction procedure, and nothing else:

Recursively Defining ‘Proofs’

- More complex proofs are built up from simpler ones by adding a step, which must either be an axiom or premise, or follow by a rule from earlier lines in the proof
- That is: something is a proof if it is an axiom or premise, or it is obtained from a proof by applying one of a finite number of rules!
- So it’s a recursive structure: A basis, iterated construction procedure, and nothing else:
  1. **Base clause**: An axiom or a premise is a proof

Recursively Defining ‘Proofs’

- More complex proofs are built up from simpler ones by adding a step, which must either be an axiom or premise, or follow by a rule from earlier lines in the proof
- That is: something is a proof if it is an axiom or premise, or it is obtained from a proof by applying one of a finite number of rules!
- So it’s a recursive structure: A basis, iterated construction procedure, and nothing else:
  1. **Base clause**: An axiom or a premise is a proof
  2. **Recursion clause**: If **Pr** is a proof, then the result of applying one of the rules of our system [more on those later!] to make **Pr** one line longer, is a proof

Recursively Defining ‘Proofs’

- More complex proofs are built up from simpler ones by adding a step, which must either be an axiom or premise, or follow by a rule from earlier lines in the proof
- That is: something is a proof if it is an axiom or premise, or it is obtained from a proof by applying one of a finite number of rules!
- So it’s a recursive structure: A basis, iterated construction procedure, and nothing else:
  1. **Base clause**: An axiom or a premise is a proof
  2. **Recursion clause**: If **Pr** is a proof, then the result of applying one of the rules of our system [more on those later!] to make **Pr** one line longer, is a proof
  3. **Closure clause**: Nothing else is a proof

3. Mathematical Induction & Recursive Definitions

c. Mathematical Induction

Mathematical Induction: Motivation

- One reason to pay attention to definition by recursion: it makes possible the proof technique of “(mathematical) induction”.

Mathematical Induction: Motivation

- One reason to pay attention to definition by recursion: it makes possible the proof technique of "(mathematical) induction".
- Don't confuse this use of 'induction' with "(scientific) induction", i.e. drawing probable inferences from observations (vs. "deduction", which draws certain inferences by reasoning alone)

Mathematical Induction: Motivation

- One reason to pay attention to definition by recursion: it makes possible the proof technique of “(mathematical) induction”.
- Don’t confuse this use of ‘induction’ with “(scientific) induction”, i.e. drawing probable inferences from observations (vs. “deduction”, which draws certain inferences by reasoning alone)
- We have inductive evidence that you’ll be doing a lot of induction!

Mathematical Induction: Motivation

- One reason to pay attention to definition by recursion: it makes possible the proof technique of "(mathematical) induction".
- Don't confuse this use of 'induction' with "(scientific) induction", i.e. drawing probable inferences from observations (vs. "deduction", which draws certain inferences by reasoning alone)
- We have inductive evidence that you'll be doing a lot of induction!
- Mathematical induction can be used in any domain where some objects can be singled out as basic, and where complex objects are built up out of simpler ones by iterating an operation.

General Pattern for mathematical induction for $\mathbb{N}$

- If you can prove two facts (called the **Base case** and the **Induction step**) about a property C, then you know it holds for every natural number:

General Pattern for mathematical induction for $\mathbb{N}$

- If you can prove two facts (called the **Base case** and the **Induction step**) about a property C, then you know it holds for every natural number:
  - **Base case**: C holds of 0. (or 1, or whatever you are starting the sequence with.)

General Pattern for mathematical induction for $\mathbb{N}$

- If you can prove two facts (called the **Base case** and the **Induction step**) about a property C, then you know it holds for every natural number:
  - **Base case**: C holds of 0. (or 1, or whatever you are starting the sequence with.)
  - **Induction Step**: Assume that C holds of some given number n. (This assumption is called the *Induction Hypothesis*).
    Using this assumption, prove that C holds of $n + 1$.

General Pattern for mathematical induction for $\mathbb{N}$

- If you can prove two facts (called the **Base case** and the **Induction step**) about a property C, then you know it holds for every natural number:
  - **Base case**: C holds of 0. (or 1, or whatever you are starting the sequence with.)
  - **Induction Step**: Assume that C holds of some given number n. (This assumption is called the *Induction Hypothesis*). Using this assumption, prove that C holds of $n + 1$.
- ALWAYS REMEMBER TO EXPLICITLY NOTE BOTH THE **BASE CASE(s)** AND THE **INDUCTION STEP**!

A Simple Example of Induction over $\mathbb{N}$

- Prove that the sum of the first n natural numbers is $\frac{1}{2}n(n+1)$ for any natural number n

A Simple Example of Induction over $\mathbb{N}$

- Prove that the sum of the first n natural numbers is $\frac{1}{2}n(n+1)$ for any natural number n
  - (start at 1 and ignore 0, because it contributes nothing to the sum)

A Simple Example of Induction over $\mathbb{N}$

- Prove that the sum of the first n natural numbers is $\frac{1}{2}n(n+1)$ for any natural number n
  - (start at 1 and ignore 0, because it contributes nothing to the sum)
- Some notation: to write a sum of a list of numbers, you need to index them to $1, 2, \ldots n$ to get a list $m_1, m_2, \ldots m_n$. Write this as:

A Simple Example of Induction over $\mathbb{N}$

- Prove that the sum of the first n natural numbers is $\frac{1}{2}n(n+1)$ for any natural number n
  - (start at 1 and ignore 0, because it contributes nothing to the sum)
- Some notation: to write a sum of a list of numbers, you need to index them to $1, 2, \ldots n$ to get a list $m_1, m_2, \ldots m_n$. Write this as:
- $\displaystyle\sum_{i=1}^{n} m_i$ (using a capital greek 'Sigma').

A Simple Example of Induction over $\mathbb{N}$

- Prove that the sum of the first n natural numbers is $\frac{1}{2}n(n+1)$ for any natural number n
  - (start at 1 and ignore 0, because it contributes nothing to the sum)
- Some notation: to write a sum of a list of numbers, you need to index them to $1, 2, \ldots n$ to get a list $m_1, m_2, \ldots m_n$. Write this as:
- $\sum_{i=1}^{n} m_i$ (using a capital greek 'Sigma').
- Now it is particularly easy to write the sum of the first n natural numbers: you don't need to index them to anything since they are in order already!

A Simple Example of Induction over $\mathbb{N}$

- Prove that the sum of the first n natural numbers is $\frac{1}{2}n(n+1)$ for any natural number n
  - (start at 1 and ignore 0, because it contributes nothing to the sum)
- Some notation: to write a sum of a list of numbers, you need to index them to $1, 2, \ldots n$ to get a list $m_1, m_2, \ldots m_n$. Write this as:
- $\sum_{i=1}^{n} m_i$ (using a capital greek 'Sigma').
- Now it is particularly easy to write the sum of the first n natural numbers: you don't need to index them to anything since they are in order already!
- So just write: $\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$

Simple Example: checking some cases

- Often helpful to orient yourself by checking some cases:

Simple Example: checking some cases

- Often helpful to orient yourself by checking some cases:
- Note that:
  $1 = \frac{1}{2}(1)(1 + 1) = \frac{2}{2} = 1$

Simple Example: checking some cases

- Often helpful to orient yourself by checking some cases:
- Note that:
  $1 = \frac{1}{2}(1)(1+1) = \frac{2}{2} = 1$
- $1 + 2 = \frac{1}{2}(2)(2+1) = 3$

Simple Example: checking some cases

- Often helpful to orient yourself by checking some cases:
- Note that:
  $1 = \frac{1}{2}(1)(1+1) = \frac{2}{2} = 1$
- $1 + 2 = \frac{1}{2}(2)(2+1) = 3$
- $1 + 2 + 3 = \frac{1}{2}(3)(3+1) = (3)(2) = 6$

Simple Example: checking some cases

- Often helpful to orient yourself by checking some cases:
- Note that:
  $1 = \frac{1}{2}(1)(1+1) = \frac{2}{2} = 1$
- $1 + 2 = \frac{1}{2}(2)(2+1) = 3$
- $1 + 2 + 3 = \frac{1}{2}(3)(3+1) = (3)(2) = 6$
- $1 + 2 + 3 + 4 = \frac{1}{2}(4)(4+1) = (2)(5) = 10$

  Each of these is an instance of the formula $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Simple Example: checking some cases

- Often helpful to orient yourself by checking some cases:
- Note that:
  $1 = \frac{1}{2}(1)(1+1) = \frac{2}{2} = 1$
- $1 + 2 = \frac{1}{2}(2)(2+1) = 3$
- $1 + 2 + 3 = \frac{1}{2}(3)(3+1) = (3)(2) = 6$
- $1 + 2 + 3 + 4 = \frac{1}{2}(4)(4+1) = (2)(5) = 10$

  Each of these is an instance of the formula $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

- When you try the formula in simple cases, it works. You might conjecture that it holds generally – but that isn't a *proof*.

Simple Example: Motivating proof by Induction

- How can we prove that this formula holds generally?

Simple Example: Motivating proof by Induction

- How can we prove that this formula holds generally?
- Note this foothold: there is a way to break down each case so that it depends straightforwardly on the immediately previous one.

Simple Example: Motivating proof by Induction

- How can we prove that this formula holds generally?
- Note this foothold: there is a way to break down each case so that it depends straightforwardly on the immediately previous one.
- Say that I ask you to compute $1 + 2 + 3 + 4 + 5$ and you already know that $1 + 2 + 3 + 4 = 10$

Simple Example: Motivating proof by Induction

- How can we prove that this formula holds generally?
- Note this foothold: there is a way to break down each case so that it depends straightforwardly on the immediately previous one.
- Say that I ask you to compute $1 + 2 + 3 + 4 + 5$ and you already know that $1 + 2 + 3 + 4 = 10$
- Note that $1 + 2 + 3 + 4 + 5 = \underbrace{[1 + 2 + 3 + 4]}_{\text{instance of prior case}} + 5$

Simple Example: Motivating proof by Induction

- How can we prove that this formula holds generally?
- Note this foothold: there is a way to break down each case so that it depends straightforwardly on the immediately previous one.
- Say that I ask you to compute $1 + 2 + 3 + 4 + 5$ and you already know that $1 + 2 + 3 + 4 = 10$
- Note that $1 + 2 + 3 + 4 + 5 = \underbrace{[1 + 2 + 3 + 4]}_{\text{instance of prior case}} + 5$
- Simplifies to $10 + 5 = 15$, given what you already know.

Simple Example: Motivating proof by Induction

- Of course, "$1 + 2 + 3 + 4 + 5$" is a small enough number that it's easy to just add the numbers together without bothering with simplifying tricks.

Simple Example: Motivating proof by Induction

- Of course, "$1 + 2 + 3 + 4 + 5$" is a small enough number that it's easy to just add the numbers together without bothering with simplifying tricks.
- But with bigger numbers that becomes less and less of an attractive option. Using the simplifying trick saves loads of time.

Simple Example: Motivating proof by Induction

- Of course, "$1 + 2 + 3 + 4 + 5$" is a small enough number that it's easy to just add the numbers together without bothering with simplifying tricks.
- But with bigger numbers that becomes less and less of an attractive option. Using the simplifying trick saves loads of time.
- It might take some time to add up the first 1,000,000 numbers, but if I already know the sum of the first 999,999, my job is easy:

Simple Example: Motivating proof by Induction

- Of course, “$1 + 2 + 3 + 4 + 5$” is a small enough number that it’s easy to just add the numbers together without bothering with simplifying tricks.
- But with bigger numbers that becomes less and less of an attractive option. Using the simplifying trick saves loads of time.
- It might take some time to add up the first 1,000,000 numbers, but if I already know the sum of the first 999,999, my job is easy:
- Just take *that* sum and add 1,000,000 to it!

An Inductive Proof of this Fact

- Now we want to **prove** the formula for the sum of the first n numbers:

An Inductive Proof of this Fact

- Now we want to **prove** the formula for the sum of the first n numbers:
- $\displaystyle\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

An Inductive Proof of this Fact

- Now we want to **prove** the formula for the sum of the first n numbers:
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- So we first need to prove the base case. In this problem, the base case is $n = 1$.

An Inductive Proof of this Fact

- Now we want to **prove** the formula for the sum of the first n numbers:
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- So we first need to prove the base case. In this problem, the base case is $n = 1$.
- So we've actually already covered the base case:

An Inductive Proof of this Fact

- Now we want to **prove** the formula for the sum of the first n numbers:
- $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- So we first need to prove the base case. In this problem, the base case is $n = 1$.
- So we've actually already covered the base case:
- **Base Case:** $1 = \frac{1}{2}(1)(1+1) = \frac{2}{2} = 1$

A Concern about the Induction Hypothesis

- For the Induction Step, we first **assume** that we already know, *for some given k*, that $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$

A Concern about the Induction Hypothesis

- For the Induction Step, we first **assume** that we already know, *for some given k*, that $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$
- We call this assumption "*The Induction Hypothesis* (IH)".

A Concern about the Induction Hypothesis

- For the Induction Step, we first **assume** that we already know, *for some given* k, that $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$
- We call this assumption "*The Induction Hypothesis* (IH)".
- At this point there may be howls from the bleachers: Isn't this just assuming what you have to prove??????

A Concern about the Induction Hypothesis

- For the Induction Step, we first **assume** that we already know, *for some given k*, that $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$
- We call this assumption "*The Induction Hypothesis* (IH)".
- At this point there may be howls from the bleachers: Isn't this just assuming what you have to prove??????
- Good question! It shows you're paying attention. But no: it isn't assuming what you have to prove

A Concern about the Induction Hypothesis

- For the Induction Step, we first **assume** that we already know, *for some given k*, that $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$
- We call this assumption "*The Induction Hypothesis* (IH)".
- At this point there may be howls from the bleachers: Isn't this just assuming what you have to prove??????
- Good question! It shows you're paying attention. But no: it isn't assuming what you have to prove
- In the IH, we could have even used 'n' rather than 'k'!

Induction Hypothesis

- What we want to prove is that the formula is true for every $n \in \mathbb{N}$. The induction hypothesis assumes we know the formula is true for some **specific**, undetermined value k.

Induction Hypothesis

- What we want to prove is that the formula is true for every $n \in \mathbb{N}$. The induction hypothesis assumes we know the formula is true for some **specific**, undetermined value k.
- We will then **use** the hypothesis to prove the case for $k + 1$.

Induction Hypothesis

- What we want to prove is that the formula is true for every $n \in \mathbb{N}$. The induction hypothesis assumes we know the formula is true for some **specific**, undetermined value k.
- We will then **use** the hypothesis to prove the case for $k + 1$.
- We then **stop** assuming the induction hypothesis, and instead conclude a conditional claim:

Induction Hypothesis

- What we want to prove is that the formula is true for every $n \in \mathbb{N}$. The induction hypothesis assumes we know the formula is true for some **specific**, undetermined value k.
- We will then **use** the hypothesis to prove the case for $k + 1$.
- We then **stop** assuming the induction hypothesis, and instead conclude a conditional claim:
- **If** the induction hypothesis is true for a number k, **then** the relevant property is also true for $k + 1$

Induction Hypothesis

- Given the induction hypothesis, $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$ we want to prove the formula holds for $k+1$, i.e. that:

Induction Hypothesis

- Given the induction hypothesis, $\sum_{i=1}^{k} i = \frac{1}{2}k(k+1)$ we want to prove the formula holds for $k+1$, i.e. that:
- $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$

  [**Note:** We assume the formula is true for some number k, and we prove, *given this assumption*, that it must be true for $k+1$ as well.]

Carrying out the Induction Step

- Our target is to show that $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$

Carrying out the Induction Step

- Our target is to show that $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$
- OK, let's remember what we noted above, that:

$$1 + 2 + \ldots + (k+1) = \underbrace{[1 + 2 + 3 + \ldots + k]}_{\text{instance of prior case}} + (k+1)$$

Carrying out the Induction Step

- Our target is to show that $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$
- OK, let's remember what we noted above, that:

$$1 + 2 + \ldots + (k+1) = \underbrace{[1 + 2 + 3 + \ldots + k]}_{\text{instance of prior case}} + (k+1)$$

- So we can simplify: $1 + 2 + 3 + \ldots + (k+1) = [\frac{1}{2}k(k+1)] + (k+1)$

Carrying out the Induction Step

- Our target is to show that $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$
- OK, let's remember what we noted above, that:

$$1 + 2 + \ldots + (k+1) = \underbrace{[1 + 2 + 3 + \ldots + k]}_{\text{instance of prior case}} + (k+1)$$

- So we can simplify: $1 + 2 + 3 + \ldots + (k+1) = [\frac{1}{2}k(k+1)] + (k+1)$
- Some algebra: $[\frac{1}{2}k(k+1)] + (k+1) = [\frac{1}{2}(k^2+k)] + [\frac{2k}{2} + \frac{2}{2}]$

Carrying out the Induction Step

- Our target is to show that $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$
- OK, let's remember what we noted above, that:

$$1 + 2 + \ldots + (k+1) = \underbrace{[1 + 2 + 3 + \ldots + k]}_{\text{instance of prior case}} + (k+1)$$

- So we can simplify: $1 + 2 + 3 + \ldots + (k+1) = [\frac{1}{2}k(k+1)] + (k+1)$
- Some algebra: $[\frac{1}{2}k(k+1)] + (k+1) = [\frac{1}{2}(k^2+k)] + [\frac{2k}{2} + \frac{2}{2}]$
  $= \frac{k^2+3k+2}{2} = \frac{(k+1)(k+2)}{2} = \frac{1}{2}(k+1)(k+2)$

Carrying out the Induction Step

- Our target is to show that $\sum_{i=1}^{k+1} i = \frac{1}{2}(k+1)(k+2)$
- OK, let's remember what we noted above, that:

$$1 + 2 + \ldots + (k+1) = \underbrace{[1 + 2 + 3 + \ldots + k]}_{\text{instance of prior case}} + (k+1)$$

- So we can simplify: $1 + 2 + 3 + \ldots + (k+1) = [\frac{1}{2}k(k+1)] + (k+1)$
- Some algebra: $[\frac{1}{2}k(k+1)] + (k+1) = [\frac{1}{2}(k^2+k)] + [\frac{2k}{2} + \frac{2}{2}]$
$= \frac{k^2+3k+2}{2} = \frac{(k+1)(k+2)}{2} = \frac{1}{2}(k+1)(k+2)$
- And that's what we wanted to prove!

Recapping the Structure of Our Inductive Proof

- Recapping our Proof Steps (remember these steps!)

Recapping the Structure of Our Inductive Proof

- Recapping our Proof Steps (remember these steps!)

  1. **Base Case**: showed that $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = 1$

Recapping the Structure of Our Inductive Proof

- Recapping our Proof Steps (remember these steps!)
  1. **Base Case**: showed that $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = 1$
  2. **Induction Step**: showed that for any k, **if** $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = k$ **then** $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = k + 1$.

Recapping the Structure of Our Inductive Proof

- Recapping our Proof Steps (remember these steps!)
  1. **Base Case**: showed that $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = 1$
  2. **Induction Step**: showed that for any k, **if** $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = k$ **then** $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ is true when $n = k+1$.
- These two facts entail that $\sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$ when $n =$ **any** $k \in \mathbb{N}$, i.e. for all $n \in \mathbb{N}$

Why does Mathematical Induction Work?

- It’s easy to see how this works: say we have a specific number.

Why does Mathematical Induction Work?

- It’s easy to see how this works: say we have a specific number.
  - To be concrete, say that I give you 325.

Why does Mathematical Induction Work?

- It's easy to see how this works: say we have a specific number.
  - To be concrete, say that I give you 325.
  - How do you know that the formula holds when $n = 325$?

Why does Mathematical Induction Work?

- It's easy to see how this works: say we have a specific number.
  - To be concrete, say that I give you 325.
  - How do you know that the formula holds when $n = 325$?
- Well, the formula holds for $n = 1$ (base case) and so by applying the induction step to the base case you know that the formula holds for $n = 2$. Applying the induction step to $n = 2$ tells you the equation holds for $n = 3$ ...

Why does Mathematical Induction Work?

- It's easy to see how this works: say we have a specific number.
  - To be concrete, say that I give you 325.
  - How do you know that the formula holds when $n = 325$?
- Well, the formula holds for $n = 1$ (base case) and so by applying the induction step to the base case you know that the formula holds for $n = 2$. Applying the induction step to $n = 2$ tells you the equation holds for $n = 3$ ...
- Applying the induction step 324 times in this fashion gives you that the equation holds for $n = 325$.

Why does Mathematical Induction Work?

- It's easy to see how this works: say we have a specific number.
    - To be concrete, say that I give you 325.
    - How do you know that the formula holds when $n = 325$?
- Well, the formula holds for $n = 1$ (base case) and so by applying the induction step to the base case you know that the formula holds for $n = 2$. Applying the induction step to $n = 2$ tells you the equation holds for $n = 3$ ...
- Applying the induction step 324 times in this fashion gives you that the equation holds for $n = 325$.
- BOOM!

Why does Mathematical Induction Work?

- It's easy to see how this works: say we have a specific number.
  - To be concrete, say that I give you 325.
  - How do you know that the formula holds when $n = 325$?
- Well, the formula holds for $n = 1$ (base case) and so by applying the induction step to the base case you know that the formula holds for $n = 2$. Applying the induction step to $n = 2$ tells you the equation holds for $n = 3 \ldots$
- Applying the induction step 324 times in this fashion gives you that the equation holds for $n = 325$.
- BOOM!
  - Feel that? That's the power of induction.

A Non-inductive Proof of the Same Fact

- Here is a quick proof that, according to legend, the great mathematician Gauss found by himself when he was 7 years old

A Non-inductive Proof of the Same Fact

- Here is a quick proof that, according to legend, the great mathematician Gauss found by himself when he was 7 years old
- Take the sum $\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$ and write it twice, one above the other, the second one in the "most to least" order, and add the series term by term. That is,

A Non-inductive Proof of the Same Fact

- Here is a quick proof that, according to legend, the great mathematician Gauss found by himself when he was 7 years old
- Take the sum $\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$ and write it twice, one above the other, the second one in the “most to least” order, and add the series term by term. That is,
- $$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$$
  $$\sum_{i=1}^{n} i = n + (n-1) + (n-2) + \ldots + 1$$
  $$\overline{\hphantom{\sum_{i=1}^{n} i = n + (n-1) + (n-2) + \ldots + 1}}$$

A Non-inductive Proof of the Same Fact

- Here is a quick proof that, according to legend, the great mathematician Gauss found by himself when he was 7 years old
- Take the sum $\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$ and write it twice, one above the other, the second one in the "most to least" order, and add the series term by term. That is,
- $$\sum_{i=1}^{n} i = 1 + 2 + 3 + \ldots + n$$
  $$\sum_{i=1}^{n} i = n + (n-1) + (n-2) + \ldots + 1$$
  $$2 \times \sum_{i=1}^{n} i = \underbrace{(n+1) + (n+1) + \ldots (n+1)}_{n \text{ terms}}$$

Non-inductive Proof Continued

- $2 \times \sum_{i=1}^{n} i = \underbrace{(n+1) + (n+1) + \ldots (n+1)}_{n \text{ terms}}$

Non-inductive Proof Continued

- $2 \times \sum_{i=1}^{n} i = \underbrace{(n+1) + (n+1) + \ldots (n+1)}_{n \text{ terms}}$
- So we get $2 \times \sum_{i=1}^{n} i = n(n+1)$ and hence $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Non-inductive Proof Continued

- $2 \times \sum_{i=1}^{n} i = \underbrace{(n+1) + (n+1) + \ldots (n+1)}_{n \text{ terms}}$
- So we get $2 \times \sum_{i=1}^{n} i = n(n+1)$ and hence $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$
- Which is what we were looking for, proved without induction

Non-inductive Proof Continued

- $2 \times \sum_{i=1}^{n} i = \underbrace{(n+1) + (n+1) + \ldots (n+1)}_{n \text{ terms}}$
- So we get $2 \times \sum_{i=1}^{n} i = n(n+1)$ and hence $\sum_{i=1}^{n} i = \dfrac{n(n+1)}{2}$
- Which is what we were looking for, proved without induction
- But notice how—in contrast to kid Gauss—induction doesn't require any special insight or inspiration!

An Example to Work Through in Class!

- An analogue of a HW Problem:

An Example to Work Through in Class!

- An analogue of a HW Problem:
- Prove that for all $n > 3$, $n \in \mathbb{N}$, $2^n < n!$

where $n! := \underbrace{(n \times (n-1) \times \ldots 3 \times 2 \times 1)}_{n\ times}$

An Example to Work Through in Class!

- An analogue of a HW Problem:
- Prove that for all $n > 3$, $n \in \mathbb{N}$, $2^n < n!$

  where $n! := \underbrace{(n \times (n-1) \times \ldots 3 \times 2 \times 1)}_{n \ times}$

- Just carry out the steps, noting each explicitly:

An Example to Work Through in Class!

- An analogue of a HW Problem:
- Prove that for all $n > 3$, $n \in \mathbb{N}$, $2^n < n!$

  where $n! := \underbrace{(n \times (n-1) \times \ldots 3 \times 2 \times 1)}_{n \text{ times}}$

- Just carry out the steps, noting each explicitly:

  1) What is the **Base Case** we need to show?

An Example to Work Through in Class!

- An analogue of a HW Problem:
- Prove that for all $n > 3$, $n \in \mathbb{N}$, $2^n < n!$

  where $n! := \underbrace{(n \times (n-1) \times \ldots 3 \times 2 \times 1)}_{n \text{ times}}$

- Just carry out the steps, noting each explicitly:

  1) What is the **Base Case** we need to show?

  2) What is our **Induction step**?

     i) Induction hypothesis?; ii) What do we need to show?

An Example to Work Through in Class!

- An analogue of a HW Problem:
- Prove that for all $n > 3$, $n \in \mathbb{N}$, $2^n < n!$

$$\text{where } n! := \underbrace{(n \times (n-1) \times \ldots 3 \times 2 \times 1)}_{n \ times}$$

- Just carry out the steps, noting each explicitly:

1) What is the **Base Case** we need to show?
2) What is our **Induction step**?
   i) Induction hypothesis?; ii) What do we need to show?
3) Just do it! (Note what you've proven)

An Example to Skip in Lecture!

- Another numerical example with the same flavor (i.e. *tasty*!)
- $\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1)$
- The general strategy for attacking this problem inductively:

  1) (**Base Case**) prove $\sum_{i=1}^{1} i^2 = \frac{1}{6}(1)(1+1)(2(1)+1)$
- Just elementary arithmetic (typically base cases are *chill*)

Another example continued

2) (**Induction Step**): assume that for a given k,

$$\sum_{i=1}^{k} i^2 = \frac{1}{6}k(k+1)(2k+1)$$

- Then set out to show that

$$\sum_{i=1}^{k+1} i^2 = \frac{1}{6}(k+1)(k+2)(2(k+1)+1)$$

- The trick is to find a way to *use* the information you are given in the induction hypothesis
  - Break down the $n = k+1$ case so it consists of the $n = k$ case plus some comparatively simple other stuff.
  - This is usually the step that requires the most thought

Working through the Deets

- Happily, when we are dealing with sums, it is easy to simplify the $n = k + 1$ case by appealing to the $n = k$ case:

$$\sum_{i=1}^{k+1} i^2 = \underbrace{1 + 2 + 3 + \ldots k}_{\text{This is } \sum_{i=1}^{k} i^2} + (k+1)^2$$

$$= [\textstyle\sum_{i=1}^{k} i^2] + (k+1)^2$$

$$= \underbrace{\frac{1}{6}k(k+1)(2k+1)}_{\text{Substituting } \frac{1}{6}k(k+1)(2k+1) \text{ for } \sum_{i=1}^{k+1} i^2} + (k+1)^2.$$

Now we have an equation that doesn't have the sum operator in it at all.

Working through more Deets

$$= \underbrace{\frac{1}{6}k(k+1)(2k+1)}_{\text{Substituting } \frac{1}{6}k(k+1)(2k+1) \text{ for } \sum_{i=1}^{k+1} i^2} + (k+1)^2.$$

- We now have a *much* simpler problem in front of us:
- Consider whether we can algebraically reduce $\frac{1}{6}k(k+1)(2k+1) + (k+1)^2$ to

  $\frac{1}{6}(k+1)(k+2)(2(k+1)+1)$.
- Spoiler alert!
- It is possible. (But I would not pay $$$ for this show)

3. Mathematical Induction & Recursive Definitions

d. Ordinary vs. Complete Induction

Ordinary Induction vs. Complete Induction

- Proofs by mathematical induction fit into two slightly different schemes, depending on how much you assume in the induction hypothesis about the cases that are less complex than the one you set out to prove.

Ordinary Induction vs. Complete Induction

- Proofs by mathematical induction fit into two slightly different schemes, depending on how much you assume in the induction hypothesis about the cases that are less complex than the one you set out to prove.
- These argument patterns are equally rigorous, but they differ a bit in their logical structure, so it is worth pointing out the difference in form explicitly.

Ordinary Induction vs. Complete Induction

- Proofs by mathematical induction fit into two slightly different schemes, depending on how much you assume in the induction hypothesis about the cases that are less complex than the one you set out to prove.
- These argument patterns are equally rigorous, but they differ a bit in their logical structure, so it is worth pointing out the difference in form explicitly.
- In the cases we have just considered, we first prove some property is true of 0 (or 1, or whatever else is the first(s) in the series)

Ordinary Induction vs. Complete Induction

- Proofs by mathematical induction fit into two slightly different schemes, depending on how much you assume in the induction hypothesis about the cases that are less complex than the one you set out to prove.
- These argument patterns are equally rigorous, but they differ a bit in their logical structure, so it is worth pointing out the difference in form explicitly.
- In the cases we have just considered, we first prove some property is true of 0 (or 1, or whatever else is the first(s) in the series)
- Second, we prove that if we assume that property is true of an arbitrary n, we can prove it holds for the $(n+1)$–th case

Complete induction (a.k.a. ‘strong induction’)

- The variation is sometimes called the method of **complete induction** (or ‘**strong induction**’).

Complete induction (a.k.a. ‘strong induction’)

- The variation is sometimes called the method of **complete induction** (or ‘**strong induction**’).
- In complete induction, we modify the induction hypothesis:

Complete induction (a.k.a. ‘strong induction’)

- The variation is sometimes called the method of **complete induction** (or ‘**strong induction**’).
- In complete induction, we modify the induction hypothesis:
  1. We prove the base case as before

Complete induction (a.k.a. ‘strong induction’)

- The variation is sometimes called the method of **complete induction** (or ‘**strong induction**’).
- In complete induction, we modify the induction hypothesis:
  1. We prove the base case as before
  2. We assume that the property holds of **every** number less than a given n (adding the assumption that n is greater than whatever number was considered in the base case)

Complete induction (a.k.a. ‘strong induction’)

- The variation is sometimes called the method of **complete induction** (or ‘**strong induction**’).
- In complete induction, we modify the induction hypothesis:
  1. We prove the base case as before
  2. We assume that the property holds of **every** number less than a given n (adding the assumption that n is greater than whatever number was considered in the base case)
- The difference is that instead of assuming the thesis *just* for the number preceding a given one, we assume it for **every** number less than the given one

Complete induction (a.k.a. ‘strong induction’)

- The variation is sometimes called the method of **complete induction** (or ‘**strong induction**’).
- In complete induction, we modify the induction hypothesis:
  1. We prove the base case as before
  2. We assume that the property holds of **every** number less than a given n (adding the assumption that n is greater than whatever number was considered in the base case)
- The difference is that instead of assuming the thesis *just* for the number preceding a given one, we assume it for **every** number less than the given one
- Each method is equally rigorous: it just happens that one form is convenient for some problems, the other for others

Ordinary (a.k.a Weak) vs. Complete (a.k.a Strong) Induction

Ordinary mathematical induction:

Base Case: $C(x)$ holds of the stuff in base clause, e.g. `0`

Induction Step: Take any case n. If $C(x)$ holds of case n, then $C(x)$ holds of $n+1$

Ordinary (a.k.a Weak) vs. Complete (a.k.a Strong) Induction

Ordinary mathematical induction:

Base Case: $C(x)$ holds of the stuff in base clause, e.g. 0

Induction Step: Take any case n. If $C(x)$ holds of case n, then $C(x)$ holds of $n+1$

Complete induction:

Base Case: $C(x)$ holds of the stuff in base clause, e.g. 'a'

Induction Step: Take any $n > 0$ [Or $n >$ the base case index(s)]. If $C(x)$ holds for **every** $x < n$, then $C(x)$ holds of n.

Ordinary (a.k.a Weak) vs. Complete (a.k.a Strong) Induction

Ordinary mathematical induction:

Base Case: $C(x)$ holds of the stuff in base clause, e.g. 0

Induction Step: Take any case n. If $C(x)$ holds of case n, then $C(x)$ holds of $n+1$

Complete induction:

Base Case: $C(x)$ holds of the stuff in base clause, e.g. 'a'

Induction Step: Take any $n > 0$ [Or $n >$ the base case index(s)]. If $C(x)$ holds for **every** $x < n$, then $C(x)$ holds of n.

- In both cases, if you can prove the Base Case and the Induction Step, you can conclude that $C(x)$ is true for every x (yee haw!)

A time to be Complete. A time to be Strong

- In particular: When dealing with strings of symbols in a language, the method of complete induction tends to be more useful.

A time to be Complete. A time to be Strong

- In particular: When dealing with strings of symbols in a language, the method of complete induction tends to be more useful.
- Illustration: you are dealing with sentences in SL, doing induction on the number of connectives in the sentence

A time to be Complete. A time to be Strong

- In particular: When dealing with strings of symbols in a language, the method of complete induction tends to be more useful.
- Illustration: you are dealing with sentences in SL, doing induction on the number of connectives in the sentence
- Say that $\mathcal{S}$ is a sentence with $n > 0$ connectives, and $\mathcal{S} = \mathcal{S}_1 \vee \mathcal{S}_2$.

A time to be Complete. A time to be Strong

- In particular: When dealing with strings of symbols in a language, the method of complete induction tends to be more useful.
- Illustration: you are dealing with sentences in SL, doing induction on the number of connectives in the sentence
- Say that $\mathcal{S}$ is a sentence with $n > 0$ connectives, and $\mathcal{S} = \mathcal{S}_1 \vee \mathcal{S}_2$.
- Then you know that both $\mathcal{S}_1$ and $\mathcal{S}_2$ have fewer connectives than $\mathcal{S}$, but you don't know precisely how many either of them has.

A time to be Complete. A time to be Strong

- In particular: When dealing with strings of symbols in a language, the method of complete induction tends to be more useful.
- Illustration: you are dealing with sentences in SL, doing induction on the number of connectives in the sentence
- Say that $\mathcal{S}$ is a sentence with $n > 0$ connectives, and $\mathcal{S} = \mathcal{S}_1 \vee \mathcal{S}_2$.
- Then you know that both $\mathcal{S}_1$ and $\mathcal{S}_2$ have fewer connectives than $\mathcal{S}$, but you don't know precisely how many either of them has.
- The n connectives in $\mathcal{S}$ could be divided up in lots of different ways between $\mathcal{S}_1$ and $\mathcal{S}_2$, so the hypothesis used in Complete Induction is more useful: it covers all the possibilities

A time to be Complete. A time to be Strong

- In particular: When dealing with strings of symbols in a language, the method of complete induction tends to be more useful.
- Illustration: you are dealing with sentences in *SL*, doing induction on the number of connectives in the sentence
- Say that $\mathcal{S}$ is a sentence with $n > 0$ connectives, and $\mathcal{S} = \mathcal{S}_1 \vee \mathcal{S}_2$.
- Then you know that both $\mathcal{S}_1$ and $\mathcal{S}_2$ have fewer connectives than $\mathcal{S}$, but you don't know precisely how many either of them has.
- The n connectives in $\mathcal{S}$ could be divided up in lots of different ways between $\mathcal{S}_1$ and $\mathcal{S}_2$, so the hypothesis used in Complete Induction is more useful: it covers all the possibilities
- Language *SL* says "stay strong!"

3. Mathematical Induction & Recursive Definitions

e. Recursion and Induction for Palindromes

Little Languages!

- One of the problems on the problem set is aimed at helping you get accustomed to doing induction/recursion on language structures

Little Languages!

- One of the problems on the problem set is aimed at helping you get accustomed to doing induction/recursion on language structures
- In this case the “language” is pretty rudimentary: the set of all strings you can make by concatenating the characters $\{a, b\}$

Little Languages!

- One of the problems on the problem set is aimed at helping you get accustomed to doing induction/recursion on language structures
- In this case the "language" is pretty rudimentary: the set of all strings you can make by concatenating the characters $\{a, b\}$
- Let's spice things up by extending this alphabet to include the great big beautiful 'c': $\{a, b, c\}$.

Little Languages!

- One of the problems on the problem set is aimed at helping you get accustomed to doing induction/recursion on language structures
- In this case the "language" is pretty rudimentary: the set of all strings you can make by concatenating the characters $\{a, b\}$
- Let's spice things up by extending this alphabet to include the great big beautiful 'c': $\{a, b, c\}$.
- This is useful for providing a simple example of complete induction

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards (ignoring spaces and punctuation). One simple example is “race car”.

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards (ignoring spaces and punctuation). One simple example is “race car”.
- In case that example went by too fast: “Yo, banana boy!” “Do geese see God?”

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards (ignoring spaces and punctuation). One simple example is “race car”.
- In case that example went by too fast: “Yo, banana boy!” “Do geese see God?”
- There are some trivial cases too: “I” is an English language palindrome, and so is “a”. Aha!

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards (ignoring spaces and punctuation). One simple example is "race car".
- In case that example went by too fast: "Yo, banana boy!" "Do geese see God?"
- There are some trivial cases too: "I" is an English language palindrome, and so is "a". Aha!
- *Let us turn now* to our rudimentary language of the set of all strings in the alphabet $\{a, b, c\}$, to study palindromes there

Palindromes semordnilaP

- A **palindrome** is a string of letters that reads the same way backwards and forwards (ignoring spaces and punctuation). One simple example is “race car”.
- In case that example went by too fast: “Yo, banana boy!” “Do geese see God?”
- There are some trivial cases too: “I” is an English language palindrome, and so is “a”. Aha!
- *Let us turn now* to our rudimentary language of the set of all strings in the alphabet $\{a, b, c\}$, to study palindromes there

  (We sometimes wonder: what if the reader said at this juncture, “No! Thou shall not turn now!” ?)

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)
- Here is an insight that makes induction an option:

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)
- Here is an insight that makes induction an option:
- If you remove the first and last letter of a palindrome, you get *another* palindrome, two letters shorter

  ($\Rightarrow$ *insert mind-blown emoji here* $\Leftarrow$)

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)
- Here is an insight that makes induction an option:
- If you remove the first and last letter of a palindrome, you get *another* palindrome, two letters shorter

  ($\Rightarrow$ *insert mind–blown emoji here* $\Leftarrow$)
- For instance: "*aabbaabbaa*" is a palindrome. If you remove the first and last '*a*' then you get "*abbaabba*".

  That's a palindrome too

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)
- Here is an insight that makes induction an option:
- If you remove the first and last letter of a palindrome, you get *another* palindrome, two letters shorter

  ($\Rightarrow$ *insert mind–blown emoji here* $\Leftarrow$)
- For instance: "*aabbaabbaa*" is a palindrome. If you remove the first and last '*a*' then you get "*abbaabba*".
  That's a palindrome too
- And so is "*bbaabb*", etc, tracing right back to "*aa*"

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)
- Here is an insight that makes induction an option:
- If you remove the first and last letter of a palindrome, you get *another* palindrome, two letters shorter

  ($\Rightarrow$ *insert mind-blown emoji here* $\Leftarrow$)
- For instance: "*aabbaabbaa*" is a palindrome. If you remove the first and last '*a*' then you get "*abbaabba*".
  That's a palindrome too
- And so is "*bbaabb*", etc, tracing right back to "*aa*"
  (tracing back even further, to the empty string 'ϵ', if we're string half-empty kinda people. 'What's an empty string?' you might say:

sthgisnI peeD (Deep Insights!)

- Say we want to prove things about palindromes (and boy, do we ever!)
- Here is an insight that makes induction an option:
- If you remove the first and last letter of a palindrome, you get *another* palindrome, two letters shorter

  ($\Rightarrow$ *insert mind-blown emoji here* $\Leftarrow$)
- For instance: "*aabbaabbaa*" is a palindrome. If you remove the first and last '*a*' then you get "*abbaabba*".
  That's a palindrome too
- And so is "*bbaabb*", etc, tracing right back to "*aa*"
  (tracing back even further, to the empty string 'ϵ', if we're string half-empty kinda people. 'What's an empty string?' you might say: oh, it's nothing)

Recursively Defining Palindromes

- This mind-blowing fact lets us recursively define "palindrome"

Recursively Defining Palindromes

- This mind-blowing fact lets us recursively define “palindrome”
- We can use the structure we’ve discovered to incorporate useful info about the set of palindromes into a recursive definition.

Recursively Defining Palindromes

- This mind–blowing fact lets us recursively define "palindrome"
- We can use the structure we've discovered to incorporate useful info about the set of palindromes into a recursive definition.
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:

Recursively Defining Palindromes

- This mind-blowing fact lets us recursively define “palindrome”
- We can use the structure we’ve discovered to incorporate useful info about the set of palindromes into a recursive definition.
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:
  1. **Base clause**: ‘a’, ‘b’, ‘c’, ‘aa’, ‘bb’, ‘cc’ are recursive palindromes

Recursively Defining Palindromes

- This mind-blowing fact lets us recursively define "palindrome"
- We can use the structure we've discovered to incorporate useful info about the set of palindromes into a recursive definition.
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:
  1. **Base clause**: 'a', 'b', 'c', 'aa', 'bb', 'cc' are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, $b * s * b$, and $c * s * c$ are recursive palindromes

Recursively Defining Palindromes

- This mind-blowing fact lets us recursively define "palindrome"
- We can use the structure we've discovered to incorporate useful info about the set of palindromes into a recursive definition.
- Call a string over $\{a, b, c\}$ a **recursive palindrome** if it satisfies the three conditions:
  1. **Base clause**: 'a', 'b', 'c', 'aa', 'bb', 'cc' are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, $b * s * b$, and $c * s * c$ are recursive palindromes
  3. **Closure clause**: Nothing else is a recursive palindrome in $\{a, b, c\}$

Induction over Palindromes

- What we want to do now is **prove** that every palindrome in $\{a, b, c\}$ is a recursive palindrome, i.e. that our recursive definition captures the intuitive phenomena

Induction over Palindromes

- What we want to do now is **prove** that every palindrome in $\{a, b, c\}$ is a recursive palindrome, i.e. that our recursive definition captures the intuitive phenomena

- We'll do induction on *the length of a string s*

Induction over Palindromes

- What we want to do now is **prove** that every palindrome in $\{a, b, c\}$ is a recursive palindrome, i.e. that our recursive definition captures the intuitive phenomena

- We'll do induction on *the length of a string s*

- Note that what follows is NOT what you're being asked to do on PS 3, problem 1(ii).

Induction over Palindromes

- What we want to do now is **prove** that every palindrome in $\{a, b, c\}$ is a recursive palindrome, i.e. that our recursive definition captures the intuitive phenomena
- We'll do induction on *the length of a string s*
- Note that what follows is NOT what you're being asked to do on PS 3, problem 1(ii).
- 1(ii) asks you to prove a specific property of a–palindromes

Base Case of our Induction

- **Base case:** Say that s is 1 or 2 symbols long. Then:

Base Case of our Induction

- **Base case:** Say that s is 1 or 2 symbols long. Then:
  i) it is one of 'a', 'b', 'c', 'aa', 'bb', or 'cc', in which case it is both a recursive palindrome and a palindrome

Base Case of our Induction

- **Base case:** Say that s is 1 or 2 symbols long. Then:

  i) it is one of 'a', 'b', 'c', 'aa', 'bb', or 'cc', in which case it is both a recursive palindrome and a palindrome

  ii) It is one of 'ab', 'ba', 'bc', 'cb', 'ac', or 'ca' in which case it is neither a palindrome nor a recursive palindrome

Base Case of our Induction

- **Base case:** Say that s is 1 or 2 symbols long. Then:

  i) it is one of ‘a’, ‘b’, ‘c’, ‘aa’, ‘bb’, or ‘cc’, in which case it is both a recursive palindrome and a palindrome

  ii) It is one of ‘ab’, ‘ba’, ‘bc’, ‘cb’, ‘ac’, or ‘ca’ in which case it is neither a palindrome nor a recursive palindrome
- (Argument for ii): None of the strings listed in ii) can be an R–palindrome because any R–palindrome that isn’t called an R–palindrome by the base case must be called an R–palindrome by the recursion clause.

Base Case of our Induction

- **Base case:** Say that s is 1 or 2 symbols long. Then:

  i) it is one of 'a', 'b', 'c', 'aa', 'bb', or 'cc', in which case it is both a recursive palindrome and a palindrome

  ii) It is one of 'ab', 'ba', 'bc', 'cb', 'ac', or 'ca' in which case it is neither a palindrome nor a recursive palindrome
- (Argument for ii): None of the strings listed in ii) can be an R–palindrome because any R–palindrome that isn't called an R–palindrome by the base case must be called an R–palindrome by the recursion clause.

  -But if s is an R–palindrome by the recursion clause, then it must contain at least three letters.)

Stating the Induction Step

- **Induction step**: *Assume* (Induction hypothesis) that every palindrome of length less than k symbols is a recursive palindrome and conversely, where $k > 2$

Stating the Induction Step

- **Induction step**: *Assume* (Induction hypothesis) that every palindrome of length less than k symbols is a recursive palindrome and conversely, where $k > 2$

- Note that the “and conversely” means we need to prove “both directions”

Stating the Induction Step

- **Induction step**: *Assume* (Induction hypothesis) that every palindrome of length less than k symbols is a recursive palindrome and conversely, where $k > 2$

- Note that the “and conversely” means we need to prove “both directions”

- You WON’T need to prove both directions on your problem set!

Need to show: s a palindrome $\Rightarrow$ s a recursive palindrome

- Let s be an arbitrary palindrome with exactly k symbols, $k > 2$

Need to show: s a palindrome $\Rightarrow$ s a recursive palindrome

- Let s be an arbitrary palindrome with exactly k symbols, $k > 2$
- Since s has three or more symbols, we can remove the first and last letter, leaving a string s'.

Need to show: s a palindrome $\Rightarrow$ s a recursive palindrome

- Let s be an arbitrary palindrome with exactly k symbols, $k > 2$
- Since s has three or more symbols, we can remove the first and last letter, leaving a string s'.
- Since s reads the same backwards and forwards, and s' comes from s by removing the first and last letter, s' reads the same backwards as forwards. That is, s' is a palindrome.

Need to show: s a palindrome $\Rightarrow$ s a recursive palindrome

- Let s be an arbitrary palindrome with exactly k symbols, $k > 2$
- Since s has three or more symbols, we can remove the first and last letter, leaving a string s'.
- Since s reads the same backwards and forwards, and s' comes from s by removing the first and last letter, s' reads the same backwards as forwards. That is, s' is a palindrome.
- Since s' is a palindrome, and it has fewer than k letters, then it is an R−palindrome by the induction hypothesis

Need to show: s a palindrome $\Rightarrow$ s a recursive palindrome

- Let s be an arbitrary palindrome with exactly k symbols, $k > 2$
- Since s has three or more symbols, we can remove the first and last letter, leaving a string s'.
- Since s reads the same backwards and forwards, and s' comes from s by removing the first and last letter, s' reads the same backwards as forwards. That is, s' is a palindrome.
- Since s' is a palindrome, and it has fewer than k letters, then it is an R−palindrome by the induction hypothesis
- Since s is a palindrome, the first and last letter must be the same, i.e. either 'a', 'b', or 'c'. So $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where as we have shown s' is an R−palindrome.

Need to show: s a palindrome $\Rightarrow$ s a recursive palindrome

- Let s be an arbitrary palindrome with exactly k symbols, $k > 2$
- Since s has three or more symbols, we can remove the first and last letter, leaving a string s'.
- Since s reads the same backwards and forwards, and s' comes from s by removing the first and last letter, s' reads the same backwards as forwards. That is, s' is a palindrome.
- Since s' is a palindrome, and it has fewer than k letters, then it is an R–palindrome by the induction hypothesis
- Since s is a palindrome, the first and last letter must be the same, i.e. either 'a', 'b', or 'c'. So $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where as we have shown s' is an R–palindrome.
- So, by the recursion clause of the definition of R–palindrome, s is an R–palindrome.

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$
- Since $k > 2$, s cannot be an R–palindrome due to the base clause, so it must be counted by the recursion clause

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$
- Since $k > 2$, s cannot be an R–palindrome due to the base clause, so it must be counted by the recursion clause
- Hence $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where s' is an R–palindrome, by the recursion clause of the definition.

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$
- Since $k > 2$, s cannot be an R–palindrome due to the base clause, so it must be counted by the recursion clause
- Hence $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where s' is an R–palindrome, by the recursion clause of the definition.
- Since s' has fewer than k symbols, it falls in the scope of the induction hypothesis, so s' is a palindrome. Since s' is a palindrome, therefore $s = a * s' * a$, or $s = b * s' * b$, and $s = c * s' * c$ are palindromes because they will also read the same backwards and forwards.

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$
- Since $k > 2$, s cannot be an R–palindrome due to the base clause, so it must be counted by the recursion clause
- Hence $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where s' is an R–palindrome, by the recursion clause of the definition.
- Since s' has fewer than k symbols, it falls in the scope of the induction hypothesis, so s' is a palindrome. Since s' is a palindrome, therefore $s = a * s' * a$, or $s = b * s' * b$, and $s = c * s' * c$ are palindromes because they will also read the same backwards and forwards.
- Since one of these is s, s is a palindrome

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$
- Since $k > 2$, s cannot be an R–palindrome due to the base clause, so it must be counted by the recursion clause
- Hence $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where s' is an R–palindrome, by the recursion clause of the definition.
- Since s' has fewer than k symbols, it falls in the scope of the induction hypothesis, so s' is a palindrome. Since s' is a palindrome, therefore $s = a * s' * a$, or $s = b * s' * b$, and $s = c * s' * c$ are palindromes because they will also read the same backwards and forwards.
- Since one of these is s, s is a palindrome
- This completes the inductive proof. And so we ask ourselves:

NTS: s a palindrome $\Leftarrow$ s a Recursive palindrome

- Let s be an arbitrary R–palindrome with exactly k symbols, $k > 2$
- Since $k > 2$, s cannot be an R–palindrome due to the base clause, so it must be counted by the recursion clause
- Hence $s = a * s' * a$, or $s = b * s' * b$, or $s = c * s' * c$, where s' is an R–palindrome, by the recursion clause of the definition.
- Since s' has fewer than k symbols, it falls in the scope of the induction hypothesis, so s' is a palindrome. Since s' is a palindrome, therefore $s = a * s' * a$, or $s = b * s' * b$, and $s = c * s' * c$ are palindromes because they will also read the same backwards and forwards.
- Since one of these is s, s is a palindrome
- This completes the inductive proof. And so we ask ourselves: Are we not pure? "No, sir!" Panama's moody Noriega brags. "It is garbage!" Irony dooms a man—a prisoner up to new era.

Remarks on the "a-palindrome" question on the HW

- Here's the problem:

Remarks on the “a-palindrome” question on the HW

- Here’s the problem:
- Call a string over $\{a, b\}$ an “a-palindrome” if it is a palindrome that has “a” as the middle letter.

Remarks on the “a–palindrome” question on the HW

- Here’s the problem:

- Call a string over $\{a, b\}$ an “a–palindrome” if it is a palindrome that has “a” as the middle letter.

- (i) Give a recursive definition of “a – palindrome”, and
  (ii) prove by induction that every a–palindrome has an even number of “b”’s.

Two things to do! (each with substeps)

- There are two things you need to do (each with substeps):

Two things to do! (each with substeps)

- There are two things you need to do (each with substeps):
  1. Give a recursive definition of “a-palindrome”. For ease of reference, say that the definition defines the “recursive a-palindromes”

Two things to do! (each with substeps)

- There are two things you need to do (each with substeps):
  1. Give a recursive definition of “a-palindrome”. For ease of reference, say that the definition defines the “recursive a-palindromes”
  2. Using induction, show that every recursive a-palindrome has an even number of b’s

Two things to do! (each with substeps)

- There are two things you need to do (each with substeps):
  1. Give a recursive definition of “a–palindrome”. For ease of reference, say that the definition defines the “recursive a–palindromes”
  2. Using induction, show that every recursive a–palindrome has an even number of b’s
- (In principle, you would also have to prove that the two definitions pick out the same set. That is, that a–palindromes are exactly the recursive a–palindromes. But I’m NOT asking you to do this as part of PS 3)

Modifying an Earlier Analogue problem

- Now step (1) is almost identical to the treatment of what are called “recursive palindromes” above. In that more complicated case, focusing on alphabet $\{a, b\}$, we’d say:

Modifying an Earlier Analogue problem

- Now step (1) is almost identical to the treatment of what are called “recursive palindromes” above. In that more complicated case, focusing on alphabet $\{a, b\}$, we’d say:
  1. **Base clause**: ‘a’, ‘b’, ‘aa’, ‘bb’ are recursive palindromes

Modifying an Earlier Analogue problem

- Now step (1) is almost identical to the treatment of what are called “recursive palindromes” above. In that more complicated case, focusing on alphabet $\{a, b\}$, we’d say:
  1. **Base clause**: ‘a’, ‘b’, ‘aa’, ‘bb’ are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, and $b * s * b$ are recursive palindromes

Modifying an Earlier Analogue problem

- Now step (1) is almost identical to the treatment of what are called "recursive palindromes" above. In that more complicated case, focusing on alphabet $\{a, b\}$, we'd say:
  1. **Base clause**: 'a', 'b', 'aa', 'bb' are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, and $b * s * b$ are recursive palindromes
  3. **Closure clause**: Nothing else is a recursive palindrome in $\{a, b\}$

Modifying an Earlier Analogue problem

- Now step (1) is almost identical to the treatment of what are called “recursive palindromes” above. In that more complicated case, focusing on alphabet $\{a, b\}$, we’d say:
  1. **Base clause**: ‘a’, ‘b’, ‘aa’, ‘bb’ are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, and $b * s * b$ are recursive palindromes
  3. **Closure clause**: Nothing else is a recursive palindrome in $\{a, b\}$
- The definition of a–palindrome is *almost identical* to this. You really just need to modify one of the clauses

Modifying an Earlier Analogue problem

- Now step (1) is almost identical to the treatment of what are called "recursive palindromes" above. In that more complicated case, focusing on alphabet $\{a, b\}$, we'd say:
  1. **Base clause**: 'a', 'b', 'aa', 'bb' are recursive palindromes
  2. **Recursion clause**: If s is a recursive palindrome, then $a * s * a$, and $b * s * b$ are recursive palindromes
  3. **Closure clause**: Nothing else is a recursive palindrome in $\{a, b\}$
- The definition of a–palindrome is *almost identical* to this. You really just need to modify one of the clauses
- Then use induction to prove the claim about a–palindromes having an even number of "b"s

Baby Stamps!

- Let's work through an analog of PS 3, problem 3

Baby Stamps!

- Let's work through an analog of PS 3, problem 3
- Baby Stamps: prove by induction that if you have (an unlimited supply of) 2¢ and 11¢ stamps, you can make any integer amount greater than or equal to 10¢

Baby Stamps!

- Let's work through an analog of PS 3, problem 3
- Baby Stamps: prove by induction that if you have (an unlimited supply of) 2¢ and 11¢ stamps, you can make any integer amount greater than or equal to 10¢

  1. Base Case(s): more complicated than usual!

Baby Stamps!

- Let's work through an analog of PS 3, problem 3
- Baby Stamps: prove by induction that if you have (an unlimited supply of) 2¢ and 11¢ stamps, you can make any integer amount greater than or equal to 10¢

1. Base Case(s): more complicated than usual!
2. Induction Step: hint—use COMPLETE induction, so consider a specific $k >$ largest base case. Then state Induction hypothesis as holding for all n less than k but ≥ 10

Hints for Question 4, PS 3

- Hint: the trick is to figure out *what* to do induction *over*!

Hints for Question 4, PS 3

- Hint: the trick is to figure out *what* to do induction *over*!

- In the base case, consider two arbitrary odd numbers, and use the hint on the PS to express them (i.e. a natural number d is odd if and only if there exists another natural number k such that $d = 2k + 1$).

Hints for Question 4, PS 3

- Hint: the trick is to figure out *what* to do induction *over*!
- In the base case, consider two arbitrary odd numbers, and use the hint on the PS to express them (i.e. a natural number d is odd if and only if there exists another natural number k such that $d = 2k + 1$).
- Use complete induction!

3. Mathematical Induction & Recursive Definitions

f. Recasting Induction in SL as Complete Induction

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property (‘**base case**’)

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
     '**Induction step**' (There are five cases to consider:)

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective '**Induction step**' (There are five cases to consider:)
     (i) $\sim\mathscr{A}$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property (‘**base case**’)
  2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective ‘**Induction step**’ (There are five cases to consider:)
     (i) $\sim\mathscr{A}$
     (ii) $(\mathscr{A} \,\&\, \mathscr{B})$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective '**Induction step**' (There are five cases to consider:)
     (i) $\sim\mathscr{A}$
     (ii) $(\mathscr{A} \,\&\, \mathscr{B})$
     (iii) $(\mathscr{A} \vee \mathscr{B})$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective
     '**Induction step**' (There are five cases to consider:)
     (i) $\sim\mathscr{A}$
     (ii) $(\mathscr{A} \,\&\, \mathscr{B})$
     (iii) $(\mathscr{A} \vee \mathscr{B})$
     (iv) $(\mathscr{A} \supset \mathscr{B})$

Special Induction Schema for the language *SL*

- Recall that for SL, we can use a special induction schema:
- If you want to prove that **ALL** sentences (wffs) of SL have a particular property, it suffices to show the following:
  1. All **atomic** sentences have that property ('**base case**')
  2. If $\mathscr{A}$ and $\mathscr{B}$ are two **arbitrary wffs** with that property, then so are the sentences built out of them by adding a connective '**Induction step**' (There are five cases to consider:)
     (i) $\sim\mathscr{A}$
     (ii) $(\mathscr{A} \,\&\, \mathscr{B})$
     (iii) $(\mathscr{A} \vee \mathscr{B})$
     (iv) $(\mathscr{A} \supset \mathscr{B})$
     (v) $(\mathscr{A} \equiv \mathscr{B})$

Complete Induction for the language *SL*

- Implicitly, this special schema is Complete Induction over the string length k of an SL sentence

Complete Induction for the language *SL*

- Implicitly, this special schema is Complete Induction over the string length k of an SL sentence
- To simplify, let's use a fragment of *SL* that contains just atomic sentences and sentences whose only connective is "&"

Complete Induction for the language *SL*

- Implicitly, this special schema is Complete Induction over the string length k of an SL sentence
- To simplify, let's use a fragment of *SL* that contains just atomic sentences and sentences whose only connective is "&"
- We can use induction because this rudimentary part of *SL* has a recursive definition:

Complete Induction for the language *SL*

- Implicitly, this special schema is Complete Induction over the string length k of an SL sentence
- To simplify, let's use a fragment of *SL* that contains just atomic sentences and sentences whose only connective is "&"
- We can use induction because this rudimentary part of *SL* has a recursive definition:
  1. **Base clause**: Every atomic sentence is a sentence of *SL*

Complete Induction for the language *SL*

- Implicitly, this special schema is Complete Induction over the string length k of an SL sentence
- To simplify, let's use a fragment of *SL* that contains just atomic sentences and sentences whose only connective is "&"
- We can use induction because this rudimentary part of *SL* has a recursive definition:
  1. **Base clause**: Every atomic sentence is a sentence of *SL*
  2. **Recursion clause**: If $\mathscr{P}$ and $\mathbb{Q}$ are sentences of *SL*, then $(\mathscr{P} \,\&\, \mathbb{Q})$ is a sentence of *SL*

Complete Induction for the language *SL*

- Implicitly, this special schema is Complete Induction over the string length k of an SL sentence
- To simplify, let's use a fragment of *SL* that contains just atomic sentences and sentences whose only connective is "&"
- We can use induction because this rudimentary part of *SL* has a recursive definition:
  1. **Base clause**: Every atomic sentence is a sentence of *SL*
  2. **Recursion clause**: If $\mathscr{P}$ and $\mathbb{Q}$ are sentences of *SL*, then $(\mathscr{P} \,\&\, \mathbb{Q})$ is a sentence of *SL*
  3. **Closure clause**: Nothing else is a sentence of (our fragment of) *SL*

- Let's reprove this claim (for our fragment): every sentence of SL has an equal number of left and right parentheses

Complete Induction for the language SL

- Let's reprove this claim (for our fragment): every sentence of SL has an equal number of left and right parentheses
- Proceed by (complete) induction on the number n of symbols in the sentence $\mathcal{S}$, i.e. the string-length:

Complete Induction for the language SL

- Let's reprove this claim (for our fragment): every sentence of SL has an equal number of left and right parentheses
- Proceed by (complete) induction on the number n of symbols in the sentence $\mathcal{S}$, i.e. the string-length:
- **Base Case**: $n = 1$

Complete Induction for the language *SL*

- Let's reprove this claim (for our fragment): every sentence of SL has an equal number of left and right parentheses
- Proceed by (complete) induction on the number n of symbols in the sentence $\mathcal{S}$, i.e. the string-length:
- **Base Case**: $n = 1$

  Then $\mathcal{S}$ is an atomic sentence, so it has no parentheses. Thus there are the same number of right and left parentheses.

Starting the Induction step

- **Induction step**: Let $\mathcal{S}$ be an arbitrary sentence of SL with k symbols, where $k > 1$. Assume that every sentence of this language with fewer than k symbols has the same number of left as right parentheses (Induction hypothesis)

Starting the Induction step

- **Induction step**: Let $\mathcal{S}$ be an arbitrary sentence of SL with k symbols, where $k > 1$. Assume that every sentence of this language with fewer than k symbols has the same number of left as right parentheses (Induction hypothesis)
- $\mathcal{S}$ is not a sentence letter, so it must be of the form $(\mathcal{S}_1 \,\&\, \mathcal{S}_2)$, with $\mathcal{S}_1$ and $\mathcal{S}_2$ sentences of SL (since we're focusing on the conjunctive fragment of SL)

Starting the Induction step

- **Induction step**: Let $\mathcal{S}$ be an arbitrary sentence of SL with k symbols, where $k > 1$. Assume that every sentence of this language with fewer than k symbols has the same number of left as right parentheses (Induction hypothesis)
- $\mathcal{S}$ is not a sentence letter, so it must be of the form $(\mathcal{S}_1 \& \mathcal{S}_2)$, with $\mathcal{S}_1$ and $\mathcal{S}_2$ sentences of SL (since we're focusing on the conjunctive fragment of SL)
- Note that $\mathcal{S}_1$ and $\mathcal{S}_2$ have fewer than k symbols

Starting the Induction step

- **Induction step**: Let $\mathcal{S}$ be an arbitrary sentence of SL with k symbols, where $k > 1$. Assume that every sentence of this language with fewer than k symbols has the same number of left as right parentheses (Induction hypothesis)
- $\mathcal{S}$ is not a sentence letter, so it must be of the form $(\mathcal{S}_1 \,\&\, \mathcal{S}_2)$, with $\mathcal{S}_1$ and $\mathcal{S}_2$ sentences of *SL* (since we're focusing on the conjunctive fragment of SL)
- Note that $\mathcal{S}_1$ and $\mathcal{S}_2$ have fewer than k symbols
- So by the induction hypothesis, both $\mathcal{S}_1$ and $\mathcal{S}_2$ have the same number of left and right parentheses

Starting the Induction step

- **Induction step**: Let $\mathcal{S}$ be an arbitrary sentence of SL with k symbols, where $k > 1$. Assume that every sentence of this language with fewer than k symbols has the same number of left as right parentheses (Induction hypothesis)
- $\mathcal{S}$ is not a sentence letter, so it must be of the form $(\mathcal{S}_1 \& \mathcal{S}_2)$, with $\mathcal{S}_1$ and $\mathcal{S}_2$ sentences of SL (since we're focusing on the conjunctive fragment of SL)
- Note that $\mathcal{S}_1$ and $\mathcal{S}_2$ have fewer than k symbols
- So by the induction hypothesis, both $\mathcal{S}_1$ and $\mathcal{S}_2$ have the same number of left and right parentheses
- (Note that we can't assign a specific length to $\mathcal{S}_1$ or $\mathcal{S}_2$. We just know that the length of each is less than k. That's why we use complete induction rather than ordinary induction in this case)

Finishing the Induction Step

- So the total number of right parentheses in $\mathcal{S}$ is:

Finishing the Induction Step

- So the total number of right parentheses in $\mathcal{S}$ is:

  1 + (number of right parentheses in $\mathcal{S}_1$) + (number of right parentheses in $\mathcal{S}_2$)

Finishing the Induction Step

- So the total number of right parentheses in $\mathcal{S}$ is:

  1 + (number of right parentheses in $\mathcal{S}_1$) + (number of right parentheses in $\mathcal{S}_2$)

- And the total number of left parentheses in $\mathcal{S}$ is:

Finishing the Induction Step

- So the total number of right parentheses in $\mathcal{S}$ is:

  1 + (number of right parentheses in $\mathcal{S}_1$) + (number of right parentheses in $\mathcal{S}_2$)

- And the total number of left parentheses in $\mathcal{S}$ is:

  1 + (number of left parentheses in $\mathcal{S}_1$) + (number of left parentheses in $\mathcal{S}_2$).

Finishing the Induction Step

- So the total number of right parentheses in $\mathcal{S}$ is:

  1 + (number of right parentheses in $\mathcal{S}_1$) + (number of right parentheses in $\mathcal{S}_2$)

- And the total number of left parentheses in $\mathcal{S}$ is:

  1 + (number of left parentheses in $\mathcal{S}_1$) + (number of left parentheses in $\mathcal{S}_2$).

- These two numbers must be equal, since they are sums of three numbers, each number equal to its counterpart in the other sum

Finishing the Induction Step

- So the total number of right parentheses in $\mathcal{S}$ is:

  1 + (number of right parentheses in $\mathcal{S}_1$) + (number of right parentheses in $\mathcal{S}_2$)

- And the total number of left parentheses in $\mathcal{S}$ is:

  1 + (number of left parentheses in $\mathcal{S}_1$) + (number of left parentheses in $\mathcal{S}_2$).

- These two numbers must be equal, since they are sums of three numbers, each number equal to its counterpart in the other sum
- This completes the induction step, and so the proof.

Remarks of a common-sensical nature

- Of course, what we just proved by induction is a piece of common sense:

Remarks of a common-sensical nature

- Of course, what we just proved by induction is a piece of common sense:
- *Of course* there will be the same number of left as right parentheses, since the only way for a parenthesis to get into a formula is to get there by an application of the recursion clause, and the recursion clause just adds exactly 1 of each type of parenthesis, every time.

Remarks of a common-sensical nature

- Of course, what we just proved by induction is a piece of common sense:
- *Of course* there will be the same number of left as right parentheses, since the only way for a parenthesis to get into a formula is to get there by an application of the recursion clause, and the recursion clause just adds exactly 1 of each type of parenthesis, every time.
- The point is not that this is a hard problem but rather that it is so simple that it helps make clear how common-sensical the inductive pattern of reasoning is, when it isn't bound up with other complications

Wise Words, from our hero Frege

- Gottlob Frege—who worked out most of the core ideas in the modern approach to logic—wrote in 1884:

Wise Words, from our hero Frege

- Gottlob Frege—who worked out most of the core ideas in the modern approach to logic—wrote in 1884:
  If, by examining the simplest cases, we can bring to light what humankind has there done by instinct, and extract from such procedures what is universally valid in them, may we not thus arrive at general methods for forming concepts and establishing principles that will be applicable in more complicated cases? (Foundations of Arithmetic §2)

Wise Words, from our hero Frege

- Gottlob Frege—who worked out most of the core ideas in the modern approach to logic—wrote in 1884:
  If, by examining the simplest cases, we can bring to light what humankind has there done by instinct, and extract from such procedures what is universally valid in them, may we not thus arrive at general methods for forming concepts and establishing principles that will be applicable in more complicated cases? (Foundations of Arithmetic §2)
- And that's what we've often done above: take common sense cases and extract a general pattern we can use in more complicated cases, where the answers are not so obvious.

Wise Words, from our hero Frege

- Gottlob Frege—who worked out most of the core ideas in the modern approach to logic—wrote in 1884:
  If, by examining the simplest cases, we can bring to light what humankind has there done by instinct, and extract from such procedures what is universally valid in them, may we not thus arrive at general methods for forming concepts and establishing principles that will be applicable in more complicated cases? (Foundations of Arithmetic §2)
- And that's what we've often done above: take common sense cases and extract a general pattern we can use in more complicated cases, where the answers are not so obvious.
- Sensing strong Descartes-energy here

3. Mathematical Induction & Recursive Definitions

g. More induction and recursion on strings

aab strings: a common sense version

- Another example of a proof by induction so simple that you might think it silly to spell it out

aab strings: a common sense version

- Another example of a proof by induction so simple that you might think it silly to spell it out
- We deal again with a “language” we know and love: all strings of letters from the alphabet $\{a, b\}$.

aab strings: a common sense version

- Another example of a proof by induction so simple that you might think it silly to spell it out
- We deal again with a “language” we know and love: all strings of letters from the alphabet $\{a, b\}$.
- Call a string in this language an ‘aab–string’ if it consists of nothing but a series of the letters aab repeated some number of times.

aab strings: a common sense version

- Another example of a proof by induction so simple that you might think it silly to spell it out
- We deal again with a “language” we know and love: all strings of letters from the alphabet $\{a, b\}$.
- Call a string in this language an ‘aab–string’ if it consists of nothing but a series of the letters aab repeated some number of times.
- So aab, $aabaab$, $aabaabaab$, $aabaabaabaab$, ... are aab–strings.

A Radical Claim

- I now claim that any aab–string has more a's in it than b's

A Radical Claim

- I now claim that any aab–string has more a's in it than b's
- Your reaction might be "Can you believe this guy? Back at it again with the trivial examples. *Of course* there are more a's than b's in an aab–string."

A Radical Claim

- I now claim that any aab–string has more a's in it than b's
- Your reaction might be "Can you believe this guy? Back at it again with the trivial examples. *Of course* there are more a's than b's in an aab–string."
- "In fact, there will always be exactly twice as many a's as b's. That's *obvious* from the definition!"

A Radical Claim

- I now claim that any aab–string has more a's in it than b's
- Your reaction might be "Can you believe this guy? Back at it again with the trivial examples. *Of course* there are more a's than b's in an aab–string."
- "In fact, there will always be exactly twice as many a's as b's. That's *obvious* from the definition!"
- When you say this, you are implicitly using recursive/inductive reasoning

Heuristic Recursive Reasoning (not a proof!)

- We spell out what is implicit in the “it’s obvious” reaction. Sometimes our common-sense reactions can have more hidden logical intricacy than we realize

Heuristic Recursive Reasoning (not a proof!)

- We spell out what is implicit in the “it’s obvious” reaction. Sometimes our common–sense reactions can have more hidden logical intricacy than we realize
- (often, things are not what they seem. Consider the dark underbelly of suburbia)

Heuristic Recursive Reasoning (not a proof!)

- We spell out what is implicit in the "it's obvious" reaction. Sometimes our common-sense reactions can have more hidden logical intricacy than we realize
- (often, things are not what they seem. Consider the dark underbelly of suburbia)
- One way to look at it is this: You get *aab* strings by repeatedly sticking *aab*'s together

Heuristic Recursive Reasoning (not a proof!)

- We spell out what is implicit in the "it's obvious" reaction. Sometimes our common-sense reactions can have more hidden logical intricacy than we realize
- (often, things are not what they seem. Consider the dark underbelly of suburbia)
- One way to look at it is this: You get aab strings by repeatedly sticking aab's together
- Each time you stick on a new aab you add one b and two a's. And you start out with more a's than b's in aab, the shortest aab-string.

Heuristic Recursive Reasoning (not a proof!)

- We spell out what is implicit in the "it's obvious" reaction. Sometimes our common–sense reactions can have more hidden logical intricacy than we realize
- (often, things are not what they seem. Consider the dark underbelly of suburbia)
- One way to look at it is this: You get aab strings by repeatedly sticking aab's together
- Each time you stick on a new aab you add one b and two a's. And you start out with more a's than b's in aab, the shortest aab–string.
- There's no way to get more b's than a's with that process, no matter how many times you repeat it.

Translating common sense into an inductive argument

- To translate the common sense reaction, we first make explicit—using recursion!—the idea that aab–strings are built up by successively adding aab:

Translating common sense into an inductive argument

- To translate the common sense reaction, we first make explicit—using recursion!—the idea that *aab*–strings are built up by successively adding *aab*:
  1. **Base clause**: *aab* is an *aab*–string

Translating common sense into an inductive argument

- To translate the common sense reaction, we first make explicit—using recursion!—the idea that aab–strings are built up by successively adding aab:
  1. **Base clause**: aab is an aab–string
  2. **Recursion clause**: If s is an aab–string, then $s * aab$ is an aab–string

Translating common sense into an inductive argument

- To translate the common sense reaction, we first make explicit—using recursion!—the idea that aab–strings are built up by successively adding aab:
    1. **Base clause**: aab is an aab–string
    2. **Recursion clause**: If s is an aab–string, then $s * aab$ is an aab–string
    3. **Closure clause**: Nothing else is an aab–string in $\{a, b\}$

Translating common sense into an inductive argument

- To translate the common sense reaction, we first make explicit—using recursion!—the idea that aab–strings are built up by successively adding aab:
  1. **Base clause**: aab is an aab–string
  2. **Recursion clause**: If s is an aab–string, then $s * aab$ is an aab–string
  3. **Closure clause**: Nothing else is an aab–string in $\{a, b\}$
- Next, use the fact that the aab–strings are built up this way to argue **by complete induction on string length** that they must have more a's than b's

Starting the Inductive Proof on aab–strings

- **Base Case**: aab has more a's than b's

Starting the Inductive Proof on aab–strings

- **Base Case**: aab has more a's than b's
- **Induction Step**: *Assume* that for any arbitrary aab–string s with fewer than k letters (where $(3 < k)$), the property holds, i.e. that this string s has more a's than b's.

Starting the Inductive Proof on aab–strings

- **Base Case**: aab has more a's than b's
- **Induction Step**: *Assume* that for any arbitrary aab–string s with fewer than k letters (where $(3 < k)$), the property holds, i.e. that this string s has more a's than b's.
  - this is our 'Induction Hypothesis' for Complete Induction

Starting the Inductive Proof on aab–strings

- **Base Case**: aab has more a's than b's
- **Induction Step**: *Assume* that for any arbitrary aab–string s with fewer than k letters (where $(3 < k)$), the property holds, i.e. that this string s has more a's than b's.
    - this is our 'Induction Hypothesis' for Complete Induction
    - $**$Note – we need to make k greater than 3, because the string in the base case has 3 letters

Starting the Inductive Proof on aab–strings

- **Base Case**: aab has more a's than b's
- **Induction Step**: *Assume* that for any arbitrary aab–string s with fewer than k letters (where $(3 < k)$), the property holds, i.e. that this string s has more a's than b's.
  - this is our 'Induction Hypothesis' for Complete Induction
  - $**$Note – we need to make k greater than 3, because the string in the base case has 3 letters
- Try to finish the proof yourself before flipping to next slide! Hint: consider an arbitrary aab–string t with k letters.

Finishing the Induction Step for aab-strings

- Let t be an arbitrary aab-string with exactly k letters

Finishing the Induction Step for aab–strings

- Let t be an arbitrary aab–string with exactly k letters
  - Since $3 < k$, that means that t cannot be aab, so it must have been produced by some applications of the recursion clause

Finishing the Induction Step for aab–strings

- Let t be an arbitrary aab–string with exactly k letters
  - Since $3 < k$, that means that t cannot be aab, so it must have been produced by some applications of the recursion clause
  - That means that there is some other aab–string t', such that $t = t' * aab$.

Finishing the Induction Step for aab–strings

- Let t be an arbitrary aab–string with exactly k letters
  - Since $3 < k$, that means that t cannot be aab, so it must have been produced by some applications of the recursion clause
  - That means that there is some other aab–string t', such that $t = t' * aab$.
- t' is three letters shorter than t so it has fewer than k letters

Finishing the Induction Step for aab–strings

- Let t be an arbitrary aab–string with exactly k letters
  - Since $3 < k$, that means that t cannot be aab, so it must have been produced by some applications of the recursion clause
  - That means that there is some other aab–string t', such that $t = t' * aab$.
- t' is three letters shorter than t so it has fewer than k letters
  - Since it has fewer than k letters it falls within the scope of the induction hypothesis, and so it has more a's than b's.

Finishing the Induction Step for aab–strings

- Let t be an arbitrary aab–string with exactly k letters
  - Since $3 < k$, that means that t cannot be aab, so it must have been produced by some applications of the recursion clause
  - That means that there is some other aab–string t', such that $t = t' * aab$.
- t' is three letters shorter than t so it has fewer than k letters
  - Since it has fewer than k letters it falls within the scope of the induction hypothesis, and so it has more a's than b's.
  - Since t has two more a's and only one more b than t', that means t has more a's than b's as well.

Finishing the Induction Step for aab–strings

- Let t be an arbitrary aab–string with exactly k letters
  - Since $3 < k$, that means that t cannot be aab, so it must have been produced by some applications of the recursion clause
  - That means that there is some other aab–string t', such that $t = t' * aab$.
- t' is three letters shorter than t so it has fewer than k letters
  - Since it has fewer than k letters it falls within the scope of the induction hypothesis, and so it has more a's than b's.
  - Since t has two more a's and only one more b than t', that means t has more a's than b's as well.
- This completes the induction step and hence the proof

3. Mathematical Induction & Recursive Definitions

h. Extra ‘Big Picture’ stuff for Recursion

Defining ‘Definition’

- What do we want a definition to do?

Defining ‘Definition’

- What do we want a definition to do?
- We want to give meaning to a word that previously had no meaning, and we want it to do it in a way that allows a person who didn’t previously know what the word meant to come to know what it means.

Defining ‘Definition’

- What do we want a definition to do?
- We want to give meaning to a word that previously had no meaning, and we want it to do it in a way that allows a person who didn’t previously know what the word meant to come to know what it means.
- Simple example: I introduce a new word into our vocabulary: grunsk. What does it mean? Well, something is grunsk if and only if it is a blue volleyball.

Defining ‘Definition’

- What do we want a definition to do?
- We want to give meaning to a word that previously had no meaning, and we want it to do it in a way that allows a person who didn’t previously know what the word meant to come to know what it means.
- Simple example: I introduce a new word into our vocabulary: grunsk. What does it mean? Well, something is grunsk if and only if it is a blue volleyball.
- OK, you might not see much point to having a word like this, but the definition does allow you to use it. All you need to now is what “ is a blue volleyball” means, and you can confidently divide the world into the grunsks and not-grunsks.

Defining ‘Definition’

- What do we want a definition to do?
- We want to give meaning to a word that previously had no meaning, and we want it to do it in a way that allows a person who didn’t previously know what the word meant to come to know what it means.
- Simple example: I introduce a new word into our vocabulary: grunsk. What does it mean? Well, something is grunsk if and only if it is a blue volleyball.
- OK, you might not see much point to having a word like this, but the definition does allow you to use it. All you need to now is what “ is a blue volleyball” means, and you can confidently divide the world into the grunsks and not-grunsks.

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.
- Another way a definition can fail is if it is circular, such as: "Something is grunsk if and only if it is a blue grunsk."

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.
- Another way a definition can fail is if it is circular, such as: "Something is grunsk if and only if it is a blue grunsk."
- That definition doesn't do what it's supposed to do.

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.
- Another way a definition can fail is if it is circular, such as: "Something is grunsk if and only if it is a blue grunsk."
- That definition doesn't do what it's supposed to do.
- Say I hand you a blue thing and ask you if it is a grunsk. You can't use the definition to decide unless you already know what a grunsk is.

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.
- Another way a definition can fail is if it is circular, such as: "Something is grunsk if and only if it is a blue grunsk."
- That definition doesn't do what it's supposed to do.
- Say I hand you a blue thing and ask you if it is a grunsk. You can't use the definition to decide unless you already know what a grunsk is.
- Let's say you hear this and you say to yourself: OK I've been warned. No circular definitions for me, thank you very much.

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.
- Another way a definition can fail is if it is circular, such as: "Something is grunsk if and only if it is a blue grunsk."
- That definition doesn't do what it's supposed to do.
- Say I hand you a blue thing and ask you if it is a grunsk. You can't use the definition to decide unless you already know what a grunsk is.
- Let's say you hear this and you say to yourself: OK I've been warned. No circular definitions for me, thank you very much.
- How to spot them so you can avoid them?

Ways to #fail: e.g. Circular Definitions

- There are ways that this pattern can screw up: it might be that the words are too vague, or perhaps they don't pick out a unique thing or class.
- Another way a definition can fail is if it is circular, such as: "Something is grunsk if and only if it is a blue grunsk."
- That definition doesn't do what it's supposed to do.
- Say I hand you a blue thing and ask you if it is a grunsk. You can't use the definition to decide unless you already know what a grunsk is.
- Let's say you hear this and you say to yourself: OK I've been warned. No circular definitions for me, thank you very much.
- How to spot them so you can avoid them?

A Guideline to Avoid Circularity

- This looks like a good guideline: set the word you want to define on a specific side of the definition (usually it's the left hand side, but our recursive definition has the word to be defined on the right). Then if the word also occurs on the *other* side, the definition is circular.

A Guideline to Avoid Circularity

- This looks like a good guideline: set the word you want to define on a specific side of the definition (usually it's the left hand side, but our recursive definition has the word to be defined on the right). Then if the word also occurs on the *other* side, the definition is circular.
- And usually, that's a trustworthy guideline. But not in the case of recursive definitions. In this case things are more subtle.

A Guideline to Avoid Circularity

- This looks like a good guideline: set the word you want to define on a specific side of the definition (usually it's the left hand side, but our recursive definition has the word to be defined on the right). Then if the word also occurs on the *other* side, the definition is circular.
- And usually, that's a trustworthy guideline. But not in the case of recursive definitions. In this case things are more subtle.
- I'll continue with the definition of our SL wffs and then revisit the point.

The Base Clause

- First the Base clause: Every atomic sentence is a wff/sentence.

The Base Clause

- First the Base clause: Every atomic sentence is a wff/sentence.
- You might think there is some circularity here: Isn't the word "sentence" on both the right and the left side, at least when we call 'wffs' 'sentences'?

The Base Clause

- First the Base clause: Every atomic sentence is a wff/sentence.
- You might think there is some circularity here: Isn't the word "sentence" on both the right and the left side, at least when we call 'wffs' 'sentences'?
- But "atomic sentence " is a defined expression: we defined it by listing the capital letters of the English alphabet, allowing subscripts from the natural numbers

The Base Clause

- First the Base clause: Every atomic sentence is a wff/sentence.
- You might think there is some circularity here: Isn't the word "sentence" on both the right and the left side, at least when we call 'wffs' 'sentences'?
- But "atomic sentence " is a defined expression: we defined it by listing the capital letters of the English alphabet, allowing subscripts from the natural numbers
- You don't need to know what the word "sentence" means to understand what an atomic sentence is from this definition.

The Whole as Distinct from its Parts

- It can happen that we understand what an expression as a whole means, even if we don't understand the parts

The Whole as Distinct from its Parts

- It can happen that we understand what an expression as a whole means, even if we don’t understand the parts
- (Most of us understand what “hoist with his own petard” means, (like many clichés, this is a turn of phrase originally introduced by Shakespeare) even though we misunderstand the relevant meaning of “hoist”, and we don’t know what a “petard” is.)

The Whole as Distinct from its Parts

- It can happen that we understand what an expression as a whole means, even if we don't understand the parts
- (Most of us understand what "hoist with his own petard" means, (like many clichés, this is a turn of phrase originally introduced by Shakespeare) even though we misunderstand the relevant meaning of "hoist", and we don't know what a "petard" is.)
- In the words of the philosopher Quine: such occurrences of words are like "cat" in "cattle" – you don't have to understand "cat" to know what cattle are.

The Whole as Distinct from its Parts

- It can happen that we understand what an expression as a whole means, even if we don't understand the parts
- (Most of us understand what "hoist with his own petard" means, (like many clichés, this is a turn of phrase originally introduced by Shakespeare) even though we misunderstand the relevant meaning of "hoist", and we don't know what a "petard" is.)
- In the words of the philosopher Quine: such occurrences of words are like "cat" in "cattle" – you don't have to understand "cat" to know what cattle are.
- You don't have to know what "sentence" means to understand the stipulation: The *atomic sentence letters* are the capital Roman letters in the English alphabet, with or without number subscripts.

3. Mathematical Induction & Recursive Definitions

i. Towers of Hanoi Example

Towers of Hanoi

- As a concrete, non-mathematical example, we'll consider a puzzle that looks at first as if it might be very hard, but it becomes simple and straightforward if we think about it in terms of recursion.

Towers of Hanoi

- As a concrete, non-mathematical example, we'll consider a puzzle that looks at first as if it might be very hard, but it becomes simple and straightforward if we think about it in terms of recursion.
- This is a tabletop puzzle called the **Towers of Hanoi**.

Towers of Hanoi

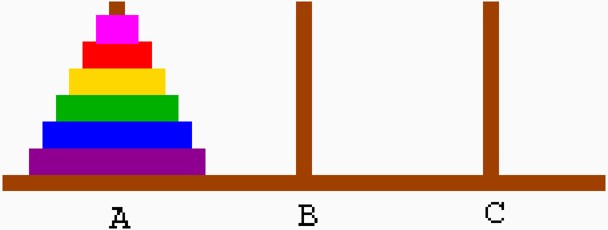
- As a concrete, non-mathematical example, we'll consider a puzzle that looks at first as if it might be very hard, but it becomes simple and straightforward if we think about it in terms of recursion.
- This is a tabletop puzzle called the **Towers of Hanoi**.
- You are given three pegs and n discs. No two discs are the same size.

Towers of Hanoi

- As a concrete, non-mathematical example, we'll consider a puzzle that looks at first as if it might be very hard, but it becomes simple and straightforward if we think about it in terms of recursion.
- This is a tabletop puzzle called the **Towers of Hanoi**.
- You are given three pegs and n discs. No two discs are the same size.
- At the beginning of the game, all n discs are all on the leftmost peg, with the largest disc on the bottom, and then the remaining discs in decreasing order of size piled on top of the first, with the smallest on top.

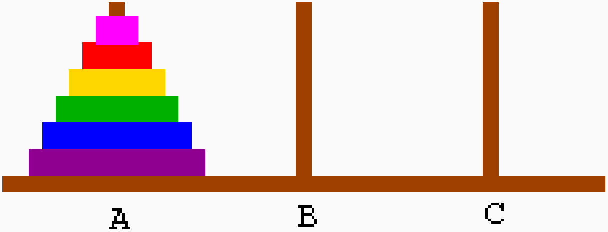
Towers of Hanoi: Initial State

- Given three pegs and *n* discs. No two discs are the same size.



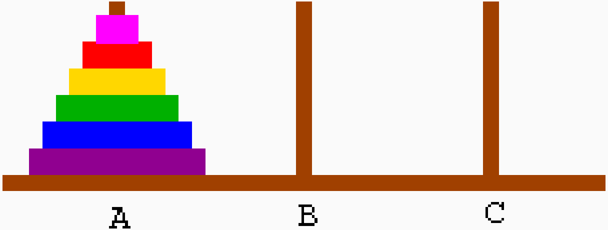
Towers of Hanoi: Initial State

- Given three pegs and n discs. No two discs are the same size.
- Initial state: all n discs are all on the leftmost peg, with the largest disc on the bottom, and then the remaining discs in decreasing order of size piled on top of the first, with the smallest on top.



Towers of Hanoi: Initial State

- Given three pegs and n discs. No two discs are the same size.
- Initial state: all n discs are all on the leftmost peg, with the largest disc on the bottom, and then the remaining discs in decreasing order of size piled on top of the first, with the smallest on top.
- Here is a drawing of the opening setup in the case of 6 discs



Towers of Hanoi: Goals and Rules

- Goal: end up with all discs on the right peg, in the same order, that is, smallest on top, and increasing in size towards the bottom.

Towers of Hanoi: Goals and Rules

- Goal: end up with all discs on the right peg, in the same order, that is, smallest on top, and increasing in size towards the bottom.
- There are restrictions on how the discs are moved:

Towers of Hanoi: Goals and Rules

- Goal: end up with all discs on the right peg, in the same order, that is, smallest on top, and increasing in size towards the bottom.
- There are restrictions on how the discs are moved:
  1. The only allowed type of move is to take **one** disc from the top of a pile and drop it on another peg.

Towers of Hanoi: Goals and Rules

- Goal: end up with all discs on the right peg, in the same order, that is, smallest on top, and increasing in size towards the bottom.
- There are restrictions on how the discs are moved:
  1. The only allowed type of move is to take **one** disc from the top of a pile and drop it on another peg.
  2. Moving several discs at once is not allowed.

Towers of Hanoi: Goals and Rules

- Goal: end up with all discs on the right peg, in the same order, that is, smallest on top, and increasing in size towards the bottom.
- There are restrictions on how the discs are moved:
  1. The only allowed type of move is to take **one** disc from the top of a pile and drop it on another peg.
  2. Moving several discs at once is not allowed.
  3. A larger disc can never lie above a smaller disc, on any peg.

Towers of Hanoi

- Question: Is it possible to solve the puzzle for any number of discs?

Towers of Hanoi

- Question: Is it possible to solve the puzzle for any number of discs?
- It turns out that by thinking recursively, we can find a very simple answer.

Towers of Hanoi

- Question: Is it possible to solve the puzzle for any number of discs?
- It turns out that by thinking recursively, we can find a very simple answer.
- If you haven't already read the notes, try experimenting with the puzzle to see what you think.

Towers of Hanoi

- Question: Is it possible to solve the puzzle for any number of discs?
- It turns out that by thinking recursively, we can find a very simple answer.
- If you haven't already read the notes, try experimenting with the puzzle to see what you think.
- There is an online version at Math is Fun!