Why are we doing this?

- Strings are **fundamental** to most formal systems (e.g. grammars).
- Strings **generalize** natural numbers (which are themselves very important) in a very natural way.
- Strings are **easy** to work with compared to set-theoretic definitions of natural numbers.
Informal Definition of Strings

- A string is just a sequence $x_1x_2\ldots x_n$ where $n \geq 0$.

- The elements of a string are called “letters” or “symbols”, but they could be members of any set.

- While infinite strings are sometimes used, for now we will be working only with finite strings.
Formal Definition of Strings

- A string over the set of letters $\Sigma$ is any member of the following inductively defined set, named $\Sigma^*$:
  - The empty string, $\varepsilon$, is in $\Sigma^*$.
  - If a string $x$ is in $\Sigma^*$, and $s$ is in $\Sigma$, then $sx$ is in $\Sigma^*$.
  - The only elements of $\Sigma^*$ are those obtained by applying the above rules.

- By $sx$ we mean a combination of $s$ with $x$ in such a way that both can be recovered from the result. This can be informally thought of as the “followed by” operator (similar to $[s| x]$ in the rex language).
Notes on the Empty String

- The empty string symbol $\lambda$ (upper-case lambda) is a “meta symbol” and as such is never an ordinary letter in $A$.

- Other symbols are often seen in place of $\lambda$:
  - $\lambda$ (lower-case lambda) is used in CS 60.
  - $\epsilon$ (lower-case epsilon) used in some texts.
Notes on $\exists x$ as an ordered-pair

- Effectively this designates an “ordered-pair” of $\exists$ with $x$, which could be written more verbosely as $(\exists, x)$.

- It is a pair because two things are combined.

- It is ordered because $(\exists, x)$ is not the same thing as $(x, \exists)$. (In fact, the latter is not meaningful in the current context.)
Ordered-Pairs as Sets

- There are ways of defining the ordered pair concept from more basic, set-theoretic, operations.

- A standard one is that every pair \((x, y)\) abbreviates a set \(\{\{x\}, \{x, y\}\}\).

- The key thing here is the assurance of the property:
  \[(x, y) = (x', y') \text{ implies } x = x' \text{ and } y = y'\]
  i.e. that pairs are decomposable into their parts.

- This can be proved.
Proof that \((x, y) = (x', y')\) implies \(x = x'\) and \(y = y'\)

- Assume \((x, y) = (x', y')\).

- So \(
\{\{x\}, \{x, y\}\} = \{\{x'\}, \{x', y'\}\}\.
\)

- If \(x = y\), then the LHS is really the same as \(
\{\{x\}\},\text{ a set of one element.}
\) So the RHS must also be a set of one element, which is to say that \(
\{x'\} = \{x', y'\}\). But this can be true only if \(x' = y'\). We have \(
\{\{x\}\} = \{\{x'\}\}.
\)
Since these are sets of one element, the elements must be equal, so \(
\{x\} = \{x'\}\). By the same argument, \(x = x'\). Combining with \(x = y\) and \(x' = y'\), we see that \(y = y'\).

- If \(x \neq y\), then the LHS is a set of two elements, and therefore the RHS is also. But \(
\{\{x'\}, \{x', y'\}\} = \text{ a set of two elements only if } x' \neq y'.
\) Since \(
\{x, y\}\) and \(
\{x', y'\}\) are now known to have two elements, while \(\{x\}\) and \(\{x'\}\)
have one, we conclude that \(x = x'\). Then we are led to conclude that \(y = y'\), for if not, the two sets of two elements could not be equal, so neither could the original two sets.
Something to Think About

- In our previous argument about pairing, we appealed to certain ideas about sets.

- How can we capture those ideas succinctly as axioms, so that we can articulate exactly what we are using about sets?
Notes on L and pairing

- L is not a pair (L is the only such string with this property).
- L could be equated to the empty set.
- Example: The string abc is constructed as pairs (a, (b, (c, L))).
- As sets, this would be:
  \[ \{\{a\}, \{a, \{\{b\}, \{b, \{\{c\}, \{c, \{\}\}\}\}\}\}\}\]  
- Now you can see why we prefer the string notation.
String Concatenation

• Concatenation means “chaining together”, i.e. following one string by another.

• Concatenation will be shown by juxtaposition: $xy$ is $y$ concatenated to $x$.

• Some texts take concatenation as a primitive, given, operation.
String Concatenation

• Concatenation is a function or binary operator on $\mathbb{N}^*$ and is defined inductively:

  • $\mathbb{N} y = y$  \hspace{1cm} \text{basis}
  • $(\mathbb{N} x)y = \mathbb{N}(xy)$  \hspace{1cm} \text{induction step}

• On the left-hand sides above is the thing we are defining, and on the right how we are defining it.

• This can be seen as analogous to defining \textit{append} in rex.
Why define by induction?

- Definition by induction is **precise**, compared to other alternatives.

- Definition by induction enables **proof** by induction.
String Axioms

• These axioms characterize strings, analogously to the way in which the Peano axioms characterize the natural numbers.

• In the following x, y, ... are implicitly quantified over strings

• ⏯, ⏯’, ... are implicitly quantified over letters.
String Axioms

- **SA1**: $(\forall x) (\text{Either } x = \emptyset \text{ or } (\forall y)(y) x = y)$
- **SA2**: $(\forall x) (\forall y) x \neq \emptyset$
- **SA3**: $(\forall x, x') (\forall y, y')$
  $(\emptyset = x' \implies \emptyset = y' \text{ and } x = x')$
- **SA4**: Let $P$ be any predicate on $\emptyset^*$. 
  
  \[ \text{basis} \quad \text{induction step} \]
  \[
  P(\emptyset), (\forall x)(P(x) \implies (\forall y)P(\emptyset x)) \quad \text{infer}
  \]
  
  $(\forall x) P(x)$

- **SA4** is a “rule of inference”, allowing to prove properties by induction.
Example of Inductive Proof

• **ST1**: $(\emptyset x) \ x\emptyset = x$, in other words, $\emptyset$ concatenated to any string is just that string.

• For proof, we identify $P(x)$ in SA4 with $x\emptyset = x$.
• SA4 says that ST1 follows if we can show two things:
  • $P(\emptyset)$, i.e. $\emptyset \emptyset = \emptyset$. **basis**
  • $(\emptyset x)(P(x) \ \emptyset (\emptyset \emptyset)P(\emptyset x))$, i.e. $(\emptyset x)(x\emptyset = x \ \emptyset (\emptyset \emptyset) (\emptyset \emptyset) = \emptyset x)$ **induction step**

• How are these two statements shown?
What we don’t need to prove

- ST1’: ($\forall x$) $\forall x = x$

This looks very similar to ST1, but we don’t need to prove it. Why?
Another Example of Inductive Proof

- ST2: \((x, y, z) \cdot x(yz) = (xy)z\). In other words, concatenation is associative.

- Here we identify \(P(x)\) in SA4 with \(x(yz) = (xy)z\).

- SA4 says that ST2 follows if we can show two things:
  - \(P(\emptyset)\), i.e. \(\emptyset(yz) = (\emptyset y)z\). \(\text{basis}\)
  - \((\emptyset x)P(x) \cup (\emptyset \emptyset)P(\emptyset x)\),
    i.e. \((\emptyset x)(x(yz) = (xy)z \cup (\emptyset \emptyset)(\emptyset x)(yz) = (((\emptyset x)y)z\) \(\text{induction step}\)

- How are these two statements shown?
Defining String Reversal

- We want to define inductively reversal ( \( R \) operator)

- Here’s an intuitively-correct idea:
  - \( \emptyset^R = \emptyset \)
  - \( (\emptyset x)^R = x^R(\emptyset) \)

- Note: We have to use \( \emptyset \) rather than just \( \emptyset \) because \( x^R\emptyset \) is not defined. But since \( \emptyset \) is a string, we can concatenate.

- Note that we are using concatenation to define this operator. This could be considered somewhat heavy-handed.
Defining String Reversal

- Another option would be to use an auxiliary function \( r \):
  - \( x^R = r(x, \square) \), where
    - \( r(\square, y) = y \) \( \quad \text{basis} \)
    - \( r(\square x, y) = r(x, \square y) \) \( \quad \text{induction step} \)

- Here we haven’t used any overt concatenation, and this makes us feel better.

- Note that this is the familiar rex definition in disguise.
- We could now try to prove that the two versions of reversal are equivalent. We’ll table this.
Proving stuff about reversal ($R$ operator)

- ST3: $(xy)^R = y^Rx^R$
- Try induction with $P(x)$: $(\overline{\overline{y}})(xy)^R = y^Rx^R$
- Basis: $P(\overline{\overline{y}})$: $(\overline{\overline{y}})(\overline{\overline{y}})^R = y^R \overline{\overline{y}}^R$
- Induction step: $(\overline{\overline{y}})(xy)^R = y^Rx^R$

implies $(\overline{\overline{y}})(\overline{\overline{y}})((\overline{\overline{x}}y)^R = y^R (\overline{\overline{x}})^R$

- LHS of $=$ is: $((\overline{\overline{x}}y)^R$
  
  $\quad = (\overline{\overline{y}}(xy))^R$
  $\quad = (xy)^R (\overline{\overline{y}})^R$
  $\quad = (y^Rx^R) (\overline{\overline{y}})^R$

- RHS of $=$ is: $y^R (\overline{\overline{x}})^R$
  
  $\quad = y^R (x^R(\overline{\overline{y}}))^R$
  $\quad = (y^Rx^R) (\overline{\overline{y}})^R$

Justify each step
Free the Monoids!

• Algebraically, $\mathfrak{S}^*$ is known as the “free monoid generated by $\mathfrak{S}$”. (usually serves to deter the casual reader from going any further)

• Recall that a monoid is a set together with:
  • An associative operator
  • An identity for the operator

• Identify the components that justify calling $\mathfrak{S}^*$ a monoid.
Natural Numbers

- Pick \( S \) to be any 1-letter alphabet. (e.g. use \( \emptyset \) as a letter).
- Then \( S^* \) is essentially the set of natural numbers:
  - \( \emptyset \) is like 0
  - \( \emptyset S \) is like \( x+1 \)
- The string axioms are then equivalent to the Peano axioms.
- The induction rule is the usual mathematical induction principle.
Natural Numbers

• Addition is just concatenation over $\mathbb{N}^*$.  

• What are subtraction, multiplication, etc.?  

• You’d need to define them inductively: more on this later.
Natural Numbers from Zip

- **0** is $\emptyset$ (basis)
- $n+1$ is $n \uparrow \{n\}$ (induction step)

for example

- 1 is $\emptyset \uparrow \{\emptyset\}$ which is $\{\emptyset\}$ which has 1 element
- 2 is $\emptyset \uparrow \{\emptyset\}$ which is $\{\emptyset, \{\emptyset\}\}$ which has 2 elements
- 3 is $\emptyset, \{\emptyset\} \uparrow \{\emptyset, \{\emptyset\}\}$ which is $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}\}$ which has 3 elements, etc.
- ...

- Let $\uparrow$ designate the set of natural numbers.
The Length | | of a String

- $|\square| = 0$  
  basis
- $|\square x| = x+1$  
  induction step
Some Properties of Length

- L1: $|xy| = |x| + |y|$
- L2: $|x^R| = |x|$
Theory of Languages

Robert M. Keller
Harvey Mudd College
2 September 2003
Definition of the Concept “Language”

- A **language** over an alphabet $\Sigma$ is any subset of $\Sigma^*$.

- Give some precise examples of languages.
Operations on Languages

- Since languages are sets, we can define their **union, intersection**, etc. just as with any sets, e.g.

  - Let L and M be two languages.

    
    \[
    L \cup M = \{ x \mid x \in L \text{ or } x \in M \} \\
    L \cap M = \{ x \mid x \in L \text{ and } x \in M \} \\
    L - M = \{ x \mid x \in L \text{ and } x \not\in M \}
    \]
Product of Languages

- Let L and M be two languages. Define

\[ LM = \{xy \mid x \in L, y \in M\} \]

called the “product” or (loosely) the “concatentation” of languages.

- Give examples.

- What if either is \( \emptyset \)?
Power Operator for Languages

- $L$ be a language. Define the “$n^{th}$ power” of $L$ inductively:

  $$L^0 = \{\varepsilon\}$$

  $$L^{n+1} = L \ L^n$$

- Examples?
Plus and Star Operators for Languages

- Let $L$ be a language. Define
  \[ L^* = L^0 \cup L^1 \cup L^2 \cup \ldots \]

- Define
  \[ L^+ = L^1 \cup L^2 \cup L^3 \cup \ldots \]

- Thus
  \[ L^* = \{\varepsilon\} \cup L^+ \]

- Give examples.
L* vs. S*

- They are the same, provided that L is the set of 1-letter strings, one for each letter in S.
Language Identities: Devise RHS’s

- $L\emptyset =$
- $L\{\square\} =$
- $(LM)N =$
- $LL^* =$
- $LL^+ =$
- $\{\square\}^* =$
- $\{\square\}^+ =$
- $\emptyset^* =$
- $\emptyset^+ =$
- $(L \quad M)N =$
- $(L \quad M^*)^* =$
- $(LM^*)^* =$
Solving a Language Equation

- This will be seen to be a useful device shortly:
- The equation \( L = LA \sqcup B \) has as a solution for \( L \):
  \[
  L = BA^*
  \]
- How can we justify this?
Regular Operators and Languages

- Union, Star, and Product (Concatenation) are called the **Regular Operators** on Languages.

- A language is **regular** if it can be formed from languages consisting of only one string of one letter each using a **finite** number of regular operators.

- Note: * counts as only one operator, despite it being defined as an infinite union.

- Examples of Regular Languages?
True or False?

- Any language of exactly one element is regular.
- Any finite language is regular.
- $\square^* - L$, where L is finite, is regular.
- Every language is regular. To see this, let $L = \{x_1, x_2, x_3, \ldots\}$.

Then $L = \{x_1\} \sqcup \{x_2\} \sqcup \{x\}_3 \sqcup \ldots$, which is clearly regular.
Regular Expressions

• A regular expression is a **shorthand** way of representing regular languages using regular operator symbols in conjunction with the following symbols.

• Each letter $s$ in $S$ stands for the language with just one string of one letter, that letter.

• $\emptyset$ stands for the language $\{\emptyset\}$.

• $\emptyset$ stands for the empty language $\emptyset$.

• Example: If $S = \{0, 1\}$, then 0 stands for the language with just one string, having one letter, 0.
Examples of Regular Expressions

- $0 \rightarrow 1$
- $(0 \rightarrow 1)^*$
- $(0 \rightarrow 1)0^*1^*$
- $((0 \rightarrow 1)0^*1)^*$
- $((\rightarrow 1)0^*1)^*$
- $0^*110^* \rightarrow 1^*001^*$
Regular Expression Notation Notes

- Instead of $\mid$, some sources use infix + or | in regular expressions.

- $\ast$ binds the tightest, then concatenation, then $\mid$.

- $\mid$ is not a regular operator, nor is -. 
Regular Expressions as Patterns

- Any language can be equated to a “pattern”, namely the pattern that matches all strings in the language.

Examples:
- 0* is the pattern that matches strings containing only 0’s
- 0*10* is the pattern that matches strings in \{0, 1\}* containing exactly one 1.
- 0*100* is the pattern that ...
- 0(0 \not\in\{1\})*1
- ((0 \not\in\{1\} (0 \not\in\{1\}))*
- (0 \not\in\{0\}) (10)* (1 \not\in\{1\})

Note: To qualify as a pattern, the language of the expression must be that of exactly the set of strings matching the pattern, not a subset or superset.
Regular Expressions as Patterns

- Give regular expressions for the following patterns over \{0, 1\}:
  - Strings in which each 1 is followed by a 0.
  - Strings in which no 1 is followed by a 0.
  - Strings in which every 1 is preceded by and followed by a 0.
  - Strings in which the number of 1’s is divisible by 3.
  - Strings in which there is no run of 3 consecutive 1’s.
Application: Searchers

- Do `man egrep` on a UNIX system.

- How do such search algorithms work?
Finite-State Automata

Robert M. Keller
Harvey Mudd College
2 September 2003
Automata

- Colloquially, an **automaton** (plural “automata”) is an autonomous device (such as a robot or wind-up toy).

- In CS, the term has a more specific meaning: that of an abstract **mathematical machine** that can perform a specific function.
Uses of Automata

- There are many uses, one of which is to specify algorithms for accepting languages.

- An automaton accepts a language if it can tell, for any given input string, whether or not the string is in the language.
Example: Compilers, etc.

- Every compiler contains an automaton, that tells whether or not the input string is well-formed, i.e. is in the language that it compiles.

- Every pattern search program is effectively an automaton for recognizing patterns.
Finite-State Automata
(FSA or DFA, they are the same)

- An automaton is finite-state if its behavior is representable by transitions between a states in a finite set, some of which are designated accepting and others not.

- Each automaton has a designated start state.
Examples of FSA

- An FSA capable of accepting exactly the strings ending with 1.

![Diagram of an FSA accepting strings ending with 1.](image-url)
Examples of FSA

- An FSA capable of accepting exactly the strings containing no two consecutive 1’s.

![FSA Diagram](image-url)
Thing to Check

- For each combination of a state and a symbol, there should be exactly one arrow leaving the state with that symbol.

- This is the “deterministic” ("D") in DFA.

- If this property does not hold, better fix it; your automaton might be wrong.
Application

- One way to implement a search is to construct, perhaps on the fly, an automaton that accepts the corresponding language, then simulate the automaton on the given input.
Two Ways to Define Specific Languages

- Give an FSA that accepts the language.
- Give a regular expression for the language.
Remarkable Fact

- The preceding two ways are equivalent.

- Equivalent here means that the two methods define the same family of languages.
Application of this Theory

- Sometimes it’s easier to give an automaton for a language.

- Sometimes it’s easier to give a regular expression.

- It would be nice to be able to go from one to the other more-or-less freely.
Regular Expression from FSA

- Label the States

- Identify each state with the set of paths from the start state to it. This set is a language.

- The language accepted by the FSA is the union of the paths to each of the accepting states, in this case L ∪ M.
Deriving Closed Forms

- View the acceptor as a set of regular-expression equations:
  - \( L = L0 \rightarrow M0 \rightarrow \)
  - \( M = L1 \)
  - \( N = M1 \rightarrow N(0 \rightarrow 1) \)
- The \( \rightarrow \) is on the RHS of the starting state only.
- We want to solve for \( L \) and \( M \), and take the union of the solutions.
Solving RE Equations

- **Solve** for L and M:
  - \( L = L0 \rightarrow M0 \rightarrow \)
  - \( M = L1 \)
  - \( N = M1 \rightarrow N(0 \rightarrow 1) \)

- **Substitution** Operation:
  - A LHS variable can be replaced with its RHS, so replacing M in the L equation:
  - \( L = L0 \rightarrow L10 \rightarrow \), or more simply
  - \( L = L(0 \rightarrow 10) \rightarrow \)

- **Elimination** Operation:
  - An equation of the form \( L = LA \rightarrow B \) has the solution \( L = BA^* \), so:
  - \( L = \rightarrow(0 \rightarrow 10)^* \), or more simply \( L = (0 \rightarrow 10)^* \)

- Substitution again:
  - \( M = L1 \)
  - \( M = (0 \rightarrow 10)^*1 \)
Conclusion

- The language accepted by the FSA below is
  - \( L \subseteq M \)
  - which is \((0 \rightarrow 10)^* \cap (0 \rightarrow 10)^*1\)
  - or more simply
  - \((0 \rightarrow 10)^*(\rightarrow \rightarrow 1)\)
FSA ↔ RE Algorithm

- Express the FSA as a set of RE equations
  - Each state is a variable.
  - Each variable is equated to a union of expressions showing how to get to that state in one step from other states.
  - The start state has $\epsilon$ on the RHS as well.

- Solve the RE equations for the variables:
  - The variables, along with their equations, are solved for one at a time.
  - Choose a variable for elimination.
  - Expression that variable in terms of the remaining variables only, using the * operator ($L = LA \epsilon B$ has the solution $L = BA^*$).
  - Substitute the solution for all occurrences of the variable in the remaining equations.
  - Repeat the above steps until no variables remain.

- Work backward, substituting the solutions found for other variables, until each variable is expressed in closed form.
Another Example

Solve:

- \( L = L_1 \cdot M_0 \cdot N_0 \cdot \cdot \cdot \)
- \( M = L_0 \cdot M_1 \cdot N_1 \)
- \( N = L_1 \cdot M_1 \cdot N_0 \)

Note that these equations don’t really correspond to a DFA, but it doesn’t matter.

Eliminate \( N \), using \( N = (L_1 \cdot M_1)0^* \)

- \( L = L_1 \cdot M_0 \cdot (L_1 \cdot M_1)0^*0 \cdot \cdot \cdot \)
- \( M = L_0 \cdot M_1 \cdot (L_1 \cdot M_1)0^*1 \)

Regroup:

- \( L = L(1 \cdot 10^*0) \cdot M(0 \cdot 10^*0) \cdot \cdot \cdot \)
- \( M = L(0 \cdot 10^*1) \cdot M(1 \cdot 10^*1) \)
Solution, continued

- Solving:
  - \( L = L(1 \rightarrow 10*0) \bowtie M(0 \rightarrow 10*0) \bowtie M \)
  - \( M = L(0 \rightarrow 10*1) \bowtie M(1 \rightarrow 10*1) \)
- Eliminate \( M \) using \( M = L(0 \rightarrow 10*1) (1 \rightarrow 10*1) \), giving:
  - \( L = L(1 \rightarrow 10*0) \bowtie L(0 \rightarrow 10*1) (1 \rightarrow 10*1)(0 \rightarrow 10*0) \bowtie M \)
- Regrouping:
  - \( L = L((1 \rightarrow 10*0) \bowtie (0 \rightarrow 10*1) (1 \rightarrow 10*1)(0 \rightarrow 10*0)) \bowtie M \)
- Solving:
  - \( L = ((1 \rightarrow 10*0) \bowtie (0 \rightarrow 10*1) (1 \rightarrow 10*1) (0 \rightarrow 10*0)) \)
- Working backward:
  - \( M = ((1 \rightarrow 10*0) \bowtie (0 \rightarrow 10*1) (1 \rightarrow 10*1) (0 \rightarrow 10*0)) \)
  - \( N = (L1 \rightarrow M1)0* = ... \)
Summary so far

- The language accepted by an FSA is a regular language.

- We haven’t shown that the converse is true.