# Theory of Automata
# Recursive Definitions

Hafiz Tayyeb Javed

Week 2

Lecture 1

# Contents

- Recursive Definition
  - Examples
- Arithmetic Expression
  - Recursive Definition of AE
- Theorems
- Propositional Calculus
  - Well Formed Formula

# Recursive definition of languages

The following three steps are used in recursive definition

1. Some basic words are specified in the language.
2. Rules for constructing more words are defined in the language.
3. No strings except those constructed in above, are allowed to be in the language.

# Example

- **Defining language of EVEN**

Step 1:

2 is in **EVEN**.

Step 2:

a. If x is in **EVEN** then x+2 and x-2 are also in **EVEN**.

b. If x and y are in **EVEN** then so are x+y, x-y and x*y.

Step 3:

No strings except those constructed in above, are allowed to be in **EVEN**.

- **Defining the language PALINDROME, defined over** $\Sigma$ = {a,b}

Step 1:

$\lambda$, $a$ and $b$ are in **PALINDROME**

Step 2:

if x is palindrome then axa, bxb, xx are also be palindrome,

Step 3:

No strings except those constructed in above, are allowed to be in palindrome

- **Defining the language $\{a^n b^n\}$, n=1,2,3,... , of strings defined over Σ={a,b}**

Step 1:

ab is in $\{\mathbf{a}^n\mathbf{b}^n\}$

Step 2:

if x is in $\{\mathbf{a}^n\mathbf{b}^n\}$, then axb is in $\{\mathbf{a}^n\mathbf{b}^n\}$

Step 3:

No strings except those constructed in above, are allowed to be in $\{\mathbf{a}^n\mathbf{b}^n\}$

- **Defining the language L, of strings ending in a , defined over $\Sigma$={a,b}**

Step 1:

a is in L

Step 2:

s(x)a is also in **L,** where s belongs to $\Sigma^*$

Step 3:

No strings except those constructed in above, are allowed to be in **L**

- **Defining the language L, of strings beginning and ending in same letters , defined over $\Sigma$={a, b}**

<u>Step 1:</u>

a and b are in L

<u>Step 2:</u>

(a)s(a) and (b)s(b) are also in **L,** where s belongs to $\Sigma^*$

<u>Step 3:</u>

No strings except those constructed in above, are allowed to be in **L**

# Arithmetic Expressions

- Suppose we ask ourselves what constitutes a valid arithmetic expression, or AE for short.
- The alphabet for this language is
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, *, /, (, )\}$

# Arithmetic Expression AE

- Obviously, the following expressions are not valid:

  (3 + 5) + 6)        2(/8 + 9)        (3 + (4-)8)

- The first contains unbalanced parentheses; the second contains the forbidden substring (/; the third contains the forbidden substring -).
- Are there more rules? The substrings // and */ are also forbidden.
- Are there still more?
- The most natural way of defining a valid AE is by using a **recursive definition**, rather than a long list of forbidden substrings.

# Recursive Definition of AE

- **Rule 1:** Any number (positive, negative, or zero) is in AE.
- **Rule 2:** If x is in AE, then so are

  *(i)* (x)

  *(ii)* -x (provided that x does not already start with a minus sign)
- **Rule 3:** If x and y are in AE, then so are

  *(i)* x + y (if the first symbol in y is not + or -)

  *(ii)* x - y (if the first symbol in y is not + or -)

  *(iii)* x * y

  *(iv)* x / y

  *(v)* x ** y (our notation for exponentiation)

- The above definition is the most natural, because it is the method we use to recognize valid arithmetic expressions in real life.
- For instance, we wish to determine if the following expression is valid:

  (2 + 4) * (7 * (9 - 3)/4)/4 * (2 + 8) - 1
- We do not really scan over the string, looking for forbidden substrings or count the parentheses.
- We actually imagine the expression in our mind broken down into components:

  Is (2 + 4) OK? Yes

  Is (9 - 3) OK? Yes

  Is 7 * (9 - 3)/4 OK? Yes, and so on.

- Note that the recursive definition of the set AE gives us the possibility of writing 8/4/2, which is ambiguous, because it could mean 8/(4/2) = 4 or (8/4)/2 = 1.
- However, the ambiguity of 8/4/2 is a problem of *meaning*. There is no doubt that this string is a word in AE, only doubt about what it means.
- By applying Rule 2, we could always put enough parentheses to avoid such a confusion.
- The recursive definition of the set AE is useful for proving many theorems about arithmetic expressions, as we shall see in the next few slides.

# Theorem 1

- **An arithmetic expression cannot contain the character $.**
- *Proof*
- This character is not part of any number, so it cannot be introduced into an AE *by Rule 1*.
- If the character string x does not contain the character $, then neither do the string (x) and -x. So, the character $ cannot be introduced into an AE *by Rule 2*.
- If neither x nor y contains the character $, then neither do any of the expressions defined in *Rule 3.*
- Therefore, the character $ can never get into an AE.

# Theorem 2 & 3

- **No arithmetic expression can begin or end with the symbol /.**
- **Proof?**

- **No arithmetic expression can contain the substring //.**
- Proof?