CS420

Introduction to the Theory of Computation

Lecture 10: Regular expressions

Tiago Cogumbreiro

Today we will learn...

- Language equivalence theorems
- summary of `Lang.v`
- Regular expressions

Example

```
Goal All >> All == All.
```

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$
- $(L \cdot \emptyset) = (\emptyset \cdot L) = \emptyset$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$
- $(L \cdot \emptyset) = (\emptyset \cdot L) = \emptyset$
- $(L \cdot \epsilon) = (\epsilon \cdot L) = (L \cup \emptyset) = (\emptyset \cup L) = (L \cup L) = L$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$
- $(L \cdot \emptyset) = (\emptyset \cdot L) = \emptyset$
- $(L \cdot \epsilon) = (\epsilon \cdot L) = (L \cup \emptyset) = (\emptyset \cup L) = (L \cup L) = L$
- $L \cup \texttt{All} = \texttt{All} \cup L = \texttt{All} = \Sigma^\star$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$
- $(L \cdot \emptyset) = (\emptyset \cdot L) = \emptyset$
- $(L \cdot \epsilon) = (\epsilon \cdot L) = (L \cup \emptyset) = (\emptyset \cup L) = (L \cup L) = L$
- $L \cup \texttt{All} = \texttt{All} \cup L = \texttt{All} = \Sigma^\star$
- $L^\star \cdot L^\star = (L^\star)^\star = L^\star$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$
- $(L \cdot \emptyset) = (\emptyset \cdot L) = \emptyset$
- $(L \cdot \epsilon) = (\epsilon \cdot L) = (L \cup \emptyset) = (\emptyset \cup L) = (L \cup L) = L$
- $L \cup \texttt{All} = \texttt{All} \cup L = \texttt{All} = \Sigma^\star$
- $L^\star \cdot L^\star = (L^\star)^\star = L^\star$
- $\epsilon^\star = \emptyset^\star = \epsilon$

Language equivalence

- $L_1 \cup (L_2 \cup L_3) = (L_1 \cup L_2) \cup L_3$
- $(L_1 \cup L_2) = (L_2 \cup L_1)$
- $(L \cdot \emptyset) = (\emptyset \cdot L) = \emptyset$
- $(L \cdot \epsilon) = (\epsilon \cdot L) = (L \cup \emptyset) = (\emptyset \cup L) = (L \cup L) = L$
- $L \cup \texttt{All} = \texttt{All} \cup L = \texttt{All} = \Sigma^\star$
- $L^\star \cdot L^\star = (L^\star)^\star = L^\star$
- $\epsilon^\star = \emptyset^\star = \epsilon$
- $(L_1 \cdot L_3) \cup (L_2 \cdot L_3) = \big((L_1 \cup L_2) \cdot L_3\big)$

Exercise

```
Goal forall (L:language),
  (All >> Void) U L == L.
Proof.
```

Lang.v examples

(Live coding...)

Regular expressions

Use Case 2

Regular Expressions: Input validation

Use Case 2

Regular Expressions: Input validation

HTML includes regular expressions to perform client-side form validation.

```
<input id="uname" name="uname" type="text"
       pattern="_([a-z]|[A-Z]|[0-9])+" minlength="4" maxlength="10">
```

- `_[a-zA-Z0-9]+`
- `[a-zA-Z0-9]` means any character beween `a` and `z`, or between `A` and `Z`, or between `0` and `9`
- `R+` means repeat `R` one or more times
- In this case, the username must start with an underscore `_`, and have one or more letters/numbers
- `minlength` and `maxlength` further restrict the string's length

Use Case 3

Regular Expressions: Text manipulation

Use Case 3

Regular Expressions: Text manipulation

Programming languages include regular expressions for fast and powerful text manipulation.

Example (JS)

```
let txt1 = "Hello World!";
let txt2 = txt1.replace(/[a-zA-Z]+/, "Bye"); // Replaces the first word by "Bye"
console.log(txt2);
// Bye World!
```

Regular expressions

A theoretical motivation

What languages can we specify with the following operators?

- `Void`
- `Nil`
- `Char`
- `App`
- `Union`
- `Star`

Regular expressions

A theoretical motivation

> What languages can we specify with the following operators?

- `Void`
- `Nil`
- `Char`
- `App`
- `Union`
- `Star`

Idea: specify a datatype that represents all possible expressions

Regular expressions

We define regular expression R as either:

Notation	***Meaning***
$\emptyset$	Rejects all words
ϵ	Only accepts the empty string
c	Only accepts a string with a single character c
$R_1 R_2$	Accepts a word from R_1 concatenated with a word from R_2
$R_1 \| \| R_2$	Accepts a word from R_1 or a word from R_2
$R^\star$	Accepts 0 or more copies of words from R

Exercise

Expression: `foo || bar`

> Is the following input accepted or rejected?

- **Input:** `[f,o,o]`

Exercise

Expression: `foo || bar`

> Is the following input accepted or rejected?

- **Input:** `[f,o,o]`
  Accept

Exercise

Expression: `foo || bar`

Is the following input accepted or rejected?

- **Input:** `[f,o,o]`
  Accept
- **Input:** `[b,a,r]`

Exercise

Expression: `foo || bar`

Is the following input accepted or rejected?

- **Input:** `[f,o,o]`
  Accept
- **Input:** `[b,a,r]`
  Accept
- **Input:** `[f,o]`

Exercise

Expression: `foo || bar`

Is the following input accepted or rejected?

- **Input:** `[f,o,o]`
  Accept
- **Input:** `[b,a,r]`
  Accept
- **Input:** `[f,o]`
  Reject
- **Input:** `[]`

Exercise

Expression: `foo || bar`

> Is the following input accepted or rejected?

- **Input:** `[f,o,o]`
  Accept
- **Input:** `[b,a,r]`
  Accept
- **Input:** `[f,o]`
  Reject
- **Input:** `[]`
  Reject

Exercise

Expression: $(\texttt{foo} \mid\mid \texttt{bar})^\star$

- **Input:** [f,o,o]

Exercise

Expression: $(\texttt{foo} || \texttt{bar})^{\star}$

- **Input:** [f,o,o]
  Accept

Exercise

Expression: $(\texttt{foo} || \texttt{bar})^\star$

- **Input:** [f,o,o]
  Accept
- **Input:** [b,a,r]

Exercise

Expression: $(\texttt{foo} \mid\mid \texttt{bar})^{\star}$

- **Input:** [f,o,o]
  Accept
- **Input:** [b,a,r]
  Accept
- **Input:** [f,o]

Exercise

Expression: $(\texttt{foo} \| \texttt{bar})^\star$

- **Input:** [f,o,o]
  Accept
- **Input:** [b,a,r]
  Accept
- **Input:** [f,o]
  Reject
- **Input:** []

Exercise

Expression: $(\texttt{foo} \mid\mid \texttt{bar})^\star$

- **Input:** [f,o,o]
  Accept
- **Input:** [b,a,r]
  Accept
- **Input:** [f,o]
  Reject
- **Input:** []
  Accept
- **Input:** [f,o,o,b,a,r]

Exercise

Expression: $(\texttt{foo} \| \texttt{bar})^\star$

- **Input:** `[f,o,o]`
  Accept
- **Input:** `[b,a,r]`
  Accept
- **Input:** `[f,o]`
  Reject
- **Input:** `[]`
  Accept
- **Input:** `[f,o,o,b,a,r]`
  Accept

Exercise

Expression

$$(\emptyset \cdot \texttt{c}) \mid\mid \texttt{aa} \mid\mid \texttt{a} \cdot \epsilon$$

- [b]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT
- [c]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT
- [c] REJECT
- [a,b]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT
- [c] REJECT
- [a,b] REJECT
- [ ]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT
- [c] REJECT
- [a,b] REJECT
- [ ] ACCEPT
- [a,a]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \mid\mid \mathtt{aa} \mid\mid \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT
- [c] REJECT
- [a,b] REJECT
- [ ] ACCEPT
- [a,a] ACCEPT
- [a]

Exercise

Expression

$$(\emptyset \cdot \mathtt{c}) \,||\, \mathtt{aa} \,||\, \mathtt{a} \cdot \epsilon$$

- [b] REJECT
- [b,c,a] REJECT
- [c] REJECT
- [a,b] REJECT
- [ ] ACCEPT
- [a,a] ACCEPT
- [a] ACCEPT

Examples

Source: `regexlib.com`

> This expression matches a hyphen separated US phone number, of the form ANN-NNN-NNNN, where A is between 2 and 9 and N is between 0 and 9.

- `[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}`

Examples

Source: `regexlib.com`

> This expression matches a hyphen separated US phone number, of the form ANN-NNN-NNNN, where A is between 2 and 9 and N is between 0 and 9.

- `[2-9][0-9]{2}-[0-9]{3}-[0-9]{4}`

Breaking it down:

- `[2-9]` corresponds to 2 `|| 3 || 4 || 5 || 7 || 8 || 9`
- `[0-9]{2}` corresponds to the power of 2, thus pattern `[0-9][0-9]`
- `-`
- `[0-9]{3}`
- `-`
- `[0-9]{4}`