Computación y Estructuras Discretas III

Andrés A. Aristizábal P.
aaaristizabal@icesi.edu.co
Ángela Villota
apvillota@icesi.edu.co

Departamento de Computación y Sistemas Inteligentes
UNIVERSIDAD ICESI

2024-2

Agenda del día

1. **Regular languages and Automata theory**
   - Introduction to Pyformlang
   - Implementing FDA with Pyformlang
   - Exercises

Agenda del día

1. **Regular languages and Automata theory**
   - Introduction to Pyformlang
   - Implementing FDA with Pyformlang
   - Exercises

What is Pyformlang?

Introduction to Pyformlang

What is Pyformlang?

- It is a Python3 library specialized in formal language manipulation.
- Designed to facilitate working with formal languages and automata theory.
- Provides a set of classes and functions that allow you to define, manipulate, and work with various types of formal languages, such as regular languages, context-free languages, and more.
- It is particularly useful for those studying theoretical computer science, formal language theory, compiler design, and related fields.
- It is composed of six modules.

Introduction to Pyformlang

Which are the modules in which Pyformlang is composed?

Introduction to Pyformlang

Which are the modules in which Pyformlang is composed?

- Regular expressions
- Finite-state automata
- Finite-state transducers
- Context-free grammars
- Push-down automata
- Indexed grammars

Which are PyFormLang main applications and use cases?

Introduction to Pyformlang

Which are PyFormLang main applications and use cases?

- Automata Theory Education:
  - To teach and learn concepts of automata theory, formal languages, and computational models in an interactive and hands-on manner.
- Language Recognition:
  - To create and manipulate finite automata for recognizing specific patterns or languages.
  - Useful for tasks like validating inputs, checking whether a given string belongs to a specific language, or implementing lexical analysis in programming languages.

Introduction to Pyformlang

- Compiler Design:
  - Automata are fundamental in compiler design for tasks like tokenization, parsing, and syntax analysis.
  - It can aid in implementing various components of a compiler.
- Natural Language Processing (NLP):
  - Some simple NLP tasks involve pattern recognition, which can be approached using finite automata.
  - It can be useful for tasks like identifying certain phrases or structures in texts.
- Regular Expressions:
  - Regular expressions can be represented using finite automata.
  - PyFormLang can help you implement and understand how regular expressions work under the hood.

Introduction to Pyformlang

- Software Verification:
  - In software engineering, automata are used for software verification and validation, ensuring that a program behaves according to its specification.
- DNA Sequence Analysis:
  - Finite automata can be applied to DNA sequence analysis, helping to identify specific patterns or sequences in genetic data.
- Code Optimization:
  - In some code optimization techniques, automata can be used to analyze code paths and identify potential optimization opportunities.
- Database Query Languages:
  - Automata can be used in designing and implementing query languages or filters for databases.

- Game Development:
  - Finite automata can be used to model and control certain behaviors in game development, such as character movement or decision-making in non-player characters.
- Pattern Matching:
  - Automata can be applied to various pattern matching problems, such as searching for specific patterns in strings or data.
- Data Validation:
  - To create automata that validate data input, ensuring that it adheres to certain rules or constraints.

1. **Regular languages and Automata theory**
   - Introduction to Pyformlang
   - Implementing FDA with Pyformlang
   - Exercises

How to start implementing a FDA?

How to start implementing a FDA?

- We import the necessary modules

```
from pyformlang.finite_automaton import
    DeterministicFiniteAutomaton, State
```

Implementing FDA with Pyformlang

- We define the states

```
q0 = State("q0")
q1 = State("q1")
```

Implementing FDA with Pyformlang

- We create the automata

```
dfa = DeterministicFiniteAutomaton(
    states={q0, q1},
    input_symbols={"0", "1"},
    start_state=q0,
    final_states={q0} # Define the set of accepting states
)
```

Implementing FDA with Pyformlang

- We add the transitions

```
dfa.add_transitions([(q0,'0',q1),
                     (q0,'1',q1),
                     (q1,'0',q0),
                     (q1,'1',q0)])
```

Implementing FDA with Pyformlang

- We test the FDA

```
print("Word '00' is accepted:", dfa.accepts('00'))
print("Word '0101' is accepted:", dfa.accepts('0101'))
print("Word '110' is not accepted:", dfa.accepts('110'))
print("Word '1' is not accepted:", dfa.accepts('1'))
```

1. **Regular languages and Automata theory**
   - Introduction to Pyformlang
   - Implementing FDA with Pyformlang
   - Exercises

Agenda del día

1. **Regular languages and Automata theory**
    - Implementing NFA with Pyformlang
    - Implementing ENFA with Pyformlang
    - Exercises

Agenda del día

1. **Regular languages and Automata theory**
   - Implementing NFA with Pyformlang
   - Implementing ENFA with Pyformlang
   - Exercises

Implementing NFA with Pyformlang

How to implement a NFA?

How to implement a NFA?

```
from pyformlang.finite_automaton import
    NondeterministicFiniteAutomaton
nfa = NondeterministicFiniteAutomaton()
nfa.add_transition('q0','a','q1')
nfa.add_transition('q0','a','q2')
nfa.add_transition('q0','c','q3')
nfa.add_transition('q1','b','q4')
nfa.add_transition('q4','a','q4')
nfa.add_transition('q2','c','q5')
nfa.add_transition('q5','c','q5')
nfa.add_transition('q3','c','q3')
nfa.add_start_state('q0')
nfa.add_final_state('q4')
nfa.add_final_state('q5')
nfa.add_final_state('q3')
```

Implementing NFA with Pyformlang

```
print(nfa.accepts('ab'))
print(nfa.accepts('abc'))
print(nfa.accepts('abaaaa'))
print(nfa.accepts('abab'))
print(nfa.accepts('accc'))
print(nfa.accepts('bac'))
print(nfa.accepts('cccc'))
```

Agenda del día

1. **Regular languages and Automata theory**
   - Implementing NFA with Pyformlang
   - Implementing ENFA with Pyformlang
   - Exercises

Implementing NFA with Pyformlang

How to implement a ENFA?

Implementing NFA with Pyformlang

How to implement a ENFA?

```
from pyformlang.finite_automaton import EpsilonNFA
enfa = EpsilonNFA()
enfa.add_transition('q0','a','q1')
enfa.add_transition('q1','epsilon','q2')
enfa.add_transition('q2','b','q2')
enfa.add_transition('q2','c','q3')
enfa.add_transition('q1','epsilon','q4')
enfa.add_transition('q4','a','q5')
enfa.add_transition('q5','a','q5')
enfa.add_transition('q1','epsilon','q6')
enfa.add_transition('q6','c','q7')
enfa.add_transition('q7','b','q8')
enfa.add_transition('q8','b','q8')
enfa.add_start_state('q0')
enfa.add_final_state('q3')
enfa.add_final_state('q5')
enfa.add_final_state('q8')
```

```python
print(enfa.accepts('ac'))
print(enfa.accepts('abc'))
print(enfa.accepts('abaaaa'))
print(enfa.accepts('abab'))
print(enfa.accepts('acbbb'))
print(enfa.accepts('bac'))
print(enfa.accepts('aaaa'))
```

Agenda del día

1. **Regular languages and Automata theory**
   - Implementing NFA with Pyformlang
   - Implementing ENFA with Pyformlang
   - Exercises