Designing Finite Automata

San Skulrattanakulchai

Feb 15, 2016

Two Design Problems

1. Given an FA, describe its language precisely.
2. Given a formal definition of a regular language, design an FA that recognizes it.

Two Design Problems

1. Given an FA, describe its language precisely.
2. Given a formal definition of a regular language, design an FA that recognizes it.

Solving Problem 1 is an art. No general principles are known.

Two Design Problems

1. Given an FA, describe its language precisely.
2. Given a formal definition of a regular language, design an FA that recognizes it.

Solving Problem 1 is an art. No general principles are known.

To solve Problem 2, work with state diagrams and follow these steps:

Two Design Problems

1. Given an FA, describe its language precisely.
2. Given a formal definition of a regular language, design an FA that recognizes it.

Solving Problem 1 is an art. No general principles are known.

To solve Problem 2, work with state diagrams and follow these steps:

1. Get all the machine states. Each state memorizes one unique machine condition.

Two Design Problems

1. Given an FA, describe its language precisely.
2. Given a formal definition of a regular language, design an FA that recognizes it.

Solving Problem 1 is an art. No general principles are known.

To solve Problem 2, work with state diagrams and follow these steps:

1. Get all the machine states. Each state memorizes one unique machine condition.

2. Pinpoint the accept states.

Two Design Problems

1. Given an FA, describe its language precisely.
2. Given a formal definition of a regular language, design an FA that recognizes it.

Solving Problem 1 is an art. No general principles are known.

To solve Problem 2, work with state diagrams and follow these steps:

1. Get all the machine states. Each state memorizes one unique machine condition.

2. Pinpoint the accept states.

3. Draw the edges.

Exercises

Assume $\Sigma = \{ \mathtt{a}, \mathtt{b} \}$. Design machines to recoqnize strings that

- starts with $\mathtt{a}$

Exercises

Assume $\Sigma = \{ \texttt{a}, \texttt{b} \}$. Design machines to recoqnize strings that

- starts with a
- ends with bb

Exercises

Assume $\Sigma = \{ \texttt{a}, \texttt{b} \}$. Design machines to recoqnize strings that

- starts with `a`
- ends with `bb`
- starts with `a` and ends with `b`

Exercises

Assume $\Sigma = \{ \texttt{a}, \texttt{b} \}$. Design machines to recoqnize strings that

- starts with `a`
- ends with `bb`
- starts with `a` and ends with `b`
- starts with `a` or ends with `a`

Exercises

Assume $\Sigma = \{ \texttt{a}, \texttt{b} \}$. Design machines to recoqnize strings that

- starts with `a`
- ends with `bb`
- starts with `a` and ends with `b`
- starts with `a` or ends with `a`
- if it starts with `a`, then it doesn't end with `a`

Exercises

- does not start or end with a but a occurs somewhere

Exercises

- does not start or end with a but a occurs somewhere
- contains ab

Exercises

- does not start or end with `a` but `a` occurs somewhere
- contains `ab`
- contains `ab` and `ba`

Exercises

- does not start or end with `a` but `a` occurs somewhere
- contains `ab`
- contains `ab` and `ba`
- contains `ab` or `ba`

Exercises

- does not start or end with `a` but `a` occurs somewhere
- contains `ab`
- contains `ab` and `ba`
- contains `ab` or `ba`
- contains neither `aa` nor `bb`

Exercises

- contains (at least) an a

Exercises

- contains (at least) an a
- contains exactly one a

Exercises

- contains (at least) an a
- contains exactly one a
- contains (at least) two a's

Exercises

- contains (at least) an a
- contains exactly one a
- contains (at least) two a's
- contains exactly two a's

Exercises

- contains (at least) an a
- contains exactly one a
- contains (at least) two a's
- contains exactly two a's
- negate each of the above

Remarks

- We can always get the complement of a language of a DFA simply by exchanging the role of the accepting and the nonaccepting states. Does this technique work for NFAs?
- The Product Construction can be used to obtain the language of binary set operations (like union, intersection, set difference, symmetric difference, etc) of two regular languages.