第 12 講 FSA と正規言語

林　恒俊

FSA が受理する言語のクラス

- 以前に正規表現が定義する言語が正規言語であることを証明している。ここでいくつかの互いに独立した言語定義法がすべて同じクラスの言語を定義していることに注意してほしい。
- FSA が定義している言語のクラスも正規言語であることが証明可能である。
- 後日の講義で正規文法という別の言語定義法が提案されるがこれも正規言語を定義する。

FSA は正規言語を定義する

- FSA が正規言語を定義することを証明するためには
  1. 任意の正規言語に対してそれを受理する FSA が存在すること
  2. 任意の FSA についてそれの定義する言語が正規言語であること

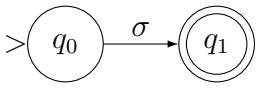
  の 2 点をいえばよい。
- あるいは正規言語と対応づけられる正規表現から証明することもできる。
  1. 任意の正規表現についてそれが定義する言語を受理する FSA が存在すること
  2. 任意の FSA についてそれの定義する言語を正規表現で定義できること

  の 2 点をいえばよい。

正規言語から FSA

- アルファベット Σ 上の正規言語からその言語を受理する FSA を構成する。例によって帰納法に基づいた手法を利用する。
- 正規言語が空集合の場合次の FSA $M_\emptyset$ を考える。

  ただし $M_\emptyset = (\{q_0, q_1\}, \Sigma, \emptyset, q_0, \{q_1\})$ で $M_\emptyset$ は遷移を持たないので受理する記号列はない。すなわち空集合を受理する。
- 正規言語が 1 個の記号の場合次の FSA M_1 を考える。



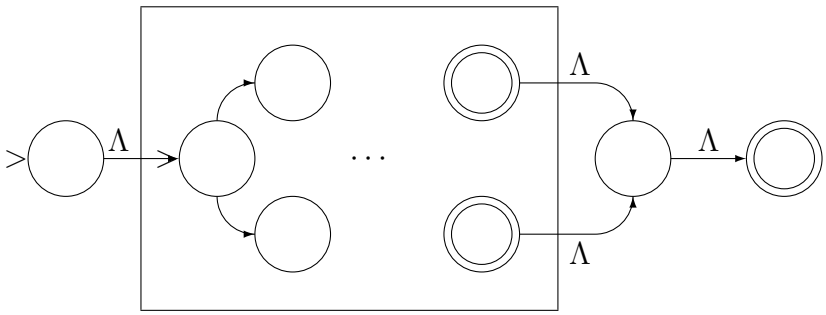
  ただし $M_1 = (\{q_0, q_1\}, \Sigma, \{(q_0, \sigma, q_1)\}, q_0, \{q_1\})$ で σ は Σ の要素である。M_1 の受理する言語は確かに $\{\sigma\}$ である。
- 前講より $\mathcal{L}_{\mathbf{FSA}}$ は和集合、連結、Kleene の $*$ 演算について閉じている。したがって正規言語 L_1, L_2 が FSA で定義可能なら $L_1 \cup L_2, L_1 \circ L_2, L_1^*$ も FSA で定義可能である。正規言語はこれらの操作のみで構成されるのですべての正規言語について対応する FSA が存在することは自明である。
- 以上で与えられた正規言語を受理する FSA が存在することが証明された。すなわち $\mathcal{L}_{\mathbf{RL}} \subseteq \mathcal{L}_{\mathbf{FSA}}$ である。
- なお正規表現から FSA を構成する証明法もある。

考察

正規表現から FSA を構成する証明で使われる手段は言語処理系の字句処理を自動的に生成する技法と全く同一である。一般に字句は正規表現を利用して定義される。したがって字句の認識はその正規表現に対応した FSA を実現すればよい。

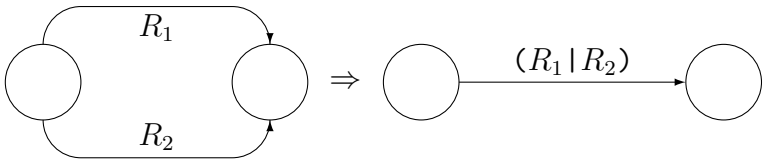
FSA から正規表現

- 与えられた FSA の受理する言語を正規表現で定義できればよい。そのためには状態遷移を正規表現で行うように拡張した FSA を利用する。このような FSA を**拡張 FSA** (**Extended FSA**, **EFA**) という。受理する言語が変化しないように EFA を変形して状態を減少させる。最終的に初期状態と終了状態を 1 個の正規表現で遷移するようにすればそれが受理言語を定義する正規表現である。
- 変換の都合上必要なら与えられた FSA に状態を追加して初期状態及び終了状態がそれぞれ 1 個づつになるようにする。



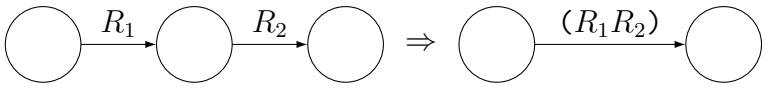
  なおこのような変更を加えても受理する言語はかわらない。
- この FSA の各遷移上の記号を正規表現とみなして EFA を構成する。そして受理する言語が変化しないように
  - 初期状態と終了状態以外の状態を段階的に取除く
  - 最終的に初期状態と 1 個の終了状態だけが残るようにする

  最終的に元の EFA が受理する言語を正規表現化したものが初期状態から終了状態への遷移に残される。
- EFA の変形は次のようになっている。なおこれらの変形を施しても受理する言語が変わらないことが理解できよう。
  - 和演算

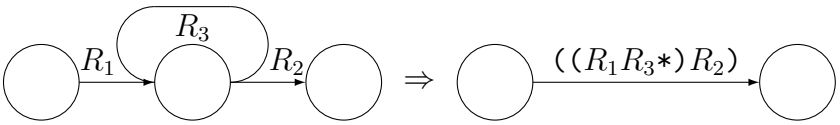


この変換では状態数は減少しないが遷移を纏めることができる。

- 連結演算



- Kleene の * 演算



この R_3 部分のない場合が連結演算である。

- EFA の初期状態及び終了状態でない状態を q とすると q を経由する遷移

$$(p)\longrightarrow(q)\longrightarrow(r)$$

について、すべての状態対 (p, r) に上記規則を適用し q を取除く。

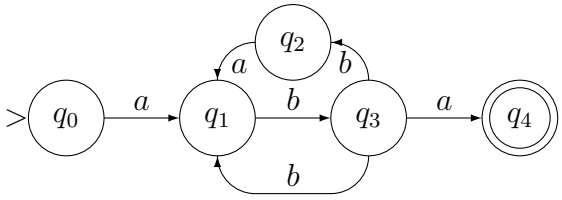
- 最終的には初期状態と終了状態が 1 個づつ残される。その間の遷移の正規表現が元の FSA が受理する言語を表現している。このようにして合成された正規表現はかなり複雑なものなることが多い。
- なお以上で定理の後半が証明された。すなわち $\mathcal{L}_{\mathbf{FSA}} \subseteq \mathcal{L}_{\mathbf{RL}}$ である。最終的に

$$\mathcal{L}_{\mathbf{FSA}} = \mathcal{L}_{\mathbf{RL}}$$

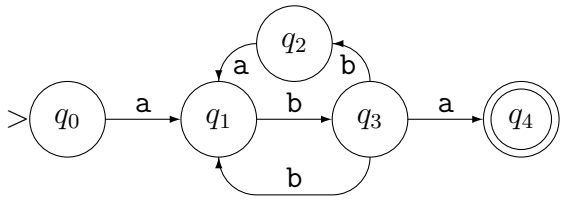
が証明された。

正規表現化例

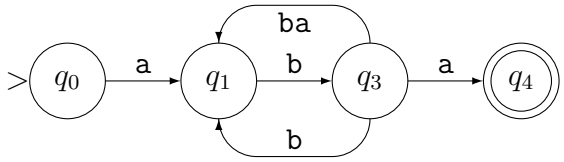
- つぎの FSA を考える。



1. これを EFA として表現する。

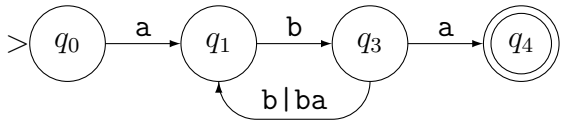


2. q_2 を除去する。



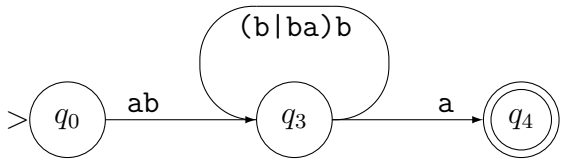
($q_3 \to q_2 \to q_1$ に連結演算則を適用)

3. 遷移を纏める。



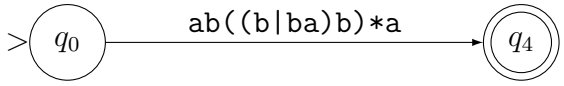
($q_3 \to q_1$ の 2 分枝に和演算則を適用)

4. q_1 を除去する。



($q_0 \to q_1 \to q_3$ と $q_3 \to q_1 \to q_3$ に連結演算則を適用)

5. Kleene の * 演算則を適用して q_3 を除去する。



- この EFA は `ab((b|ba)b)*a` または `ab(bb|bab)*a` を受理する。

考察

任意の計算機プログラムが構造化可能であることについてもこの証明技法を適用できる。非構造化プログラムは丁度状態遷移図と同様に網状に構成されている。実行文は遷移につけられたラベルに相当する。プログラムを実行したときの文の列は有限状態機械が受理する記号列になる。

上記証明により実行文の列は正規表現で表現され、正規表現から構造化プログラムを得ることが可能である。なお厳密な証明にはより深い検討が必要である。

正規言語の性質

- この結果正規言語が持ついくつかの性質について FSA を利用して検討することが可能になった。
- 例えば正規言語の補集合、逆列言語、結びなどすべて正規言語である。
- 正規言語でない言語を見極める手段を得ることができる。