确定有限状态自动机

维基百科，自由的百科全书

在计算理论中，**确定有限状态自动机**或**确定有限自动机**（英語：deterministic finite automaton, DFA）是一个能实现状态转移的自动机。对于一个给定的属于该自动机的状态和一个属于该自动机字母表Σ的字符，它都能根据事先给定的转移函数转移到下一个状态（这个状态可以是先前那个状态）。

目录

基础概念

定义

确定有限状态自动机$\mathcal{A}$是由

- 一个非空有限的状态集合Q

- 一个输入字母表Σ（非空有限的字符集合）
- 一个转移函数$\delta : Q \times \Sigma \to Q$（例如：$\delta\left(q, \sigma\right) = p, \left(p, q \in Q, \sigma \in \Sigma\right)$）
- 一个开始状态$s \in Q$
- 一个接受状态的集合$F \subseteq Q$

所组成的5-元组。因此一个DFA可以写成这样的形式：$\mathcal{A} = \left(Q, \Sigma, \delta, s, F\right)$。

工作方式（非正式的语义）

确定有限状态自动机从起始状态开始，一个字符接一个字符地读入一个字符串$w \in \Sigma^{*}$（这里的*指示Kleene星号算子。），并根据给定的转移函数一步一步地转移至下一个状态。在读完该字符串后，如果该自动机停在一个属于F的接受状态，那么它就接受该字符串，反之则拒绝该字符串。

扩展转移函数

为了在保证严谨的前提下，方便地叙述关于DFA的内容，我们定义如下扩展的转移函数：

$\delta^{*} : Q \times \Sigma^{*} \to Q$。

- $\delta^{*}\left(q, w\right)$是自动机从状态q顺序读入字符串w后达到的状态
- 扩展转移函数递归的定义为：
  - $\delta^{*}\left(q, \epsilon\right) = q$
  - $\delta^{*}\left(q, u\sigma\right) = \delta(\delta^{*}(q, u), \sigma), \forall u \in \Sigma^{*}, \sigma \in \Sigma$

工作方式（正式的语义）

对于一个确定有限状态自动机$\mathcal{A} = \left(Q, \Sigma, \delta, s, F\right)$，如果$\delta^{*}\left(s, w\right) \in F$，我们就说该自动机接受字符串w，反之则表明该自动机拒绝字符串w。

被一个确定有限自动机接受的语言（或者叫“被识别的语言”）定义为：$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^{*} | \mathcal{A}$ 接受字符串 $w\}$，也就是由所有被接受的字符串组成的集合。

DFA与有向图

除了数学上的严谨表述，通常为了讨论方便，也使用状态图直观地表示DFA。不难发现，对于一个给定的DFA，存在唯一一个对应的有向图（但是严格意义上一个有向图不能确定出唯一一个DFA）。有向图的每个结点对应一个状态，每条有向边对应一种转移。习惯上将结点画成两个圈表示接受状态，一个圈表示拒绝状态。用一条没有起点的边指向起始状态。

除了在表述上方便以外，在研究某些问题（如“给定的DFA的语言是否为无穷集合”）时，状态图也提供了有效的解法。

利弊

DFA是一種实际的计算模型，因为有平凡的线性时间、恒定空间的在线算法模拟在输入流上的DFA。给定两个DFA有有效算法找到识别它们所识别语言的并集、交集和补集的DFA。还有有效算法确定一个DFA是否接受任何给定字符串，一个DFA是否接受所有字符串，两个DFA是否识别同样的语言，和对特定正则语言找到状态数目最小的DFA（最小DFA）。

在另一方面，DFA在可识别的语言上有严格的限制—很多简单的语言，包括需要多于恒定空间来解决的任何问题，不能被DFA识别。经典的DFA不能识别的简单语言的例子是括号语言，就是由正确配对的括号组成的语言，比如 (()())。由形如$a^n b^n$的字符串组成的语言，就是有限数目个a，随后是相等数目个b。可以证明没有DFA有足够状态来识别这种语言（通俗地说，因为需要至少2n个状态，而n是不恒定的）。

其它

1. 能被确定有限状态自动机识别的语言是正则语言。
2. 确定有限状态自动机是非确定有限状态自动机的一种极限形式。
3. 确定有限状态自动机在计算能力上等价于非确定有限状态自动机。
4. 没有接受状态列表并没有指定开始状态的确定有限状态机叫做转移系统或半自动机。

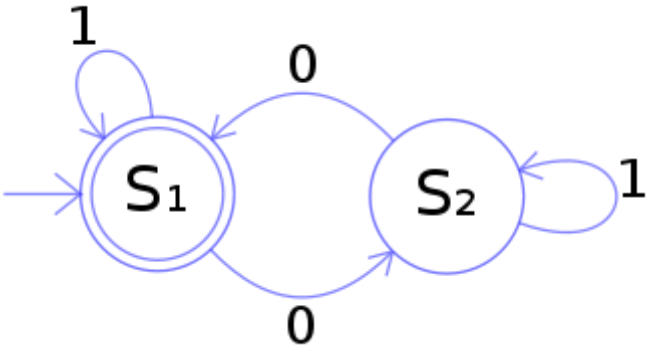
例子

下面是一个确定有限状态自动机的例子。

确定有限状态自动机$\mathcal{A} = (Q, \Sigma, \delta, s, F)$

- $Q = \{S_1, S_2\}$
- $\Sigma = \{0, 1\}$
- $s = S_1$
- $F = \{S_1\}$
- δ由下面的状态转移表定义：

	0	**1**
S_1	S_2	S_1
S_2	S_1	S_2



$\mathcal{A}$的状态图

- 对应的转移函数为：
  - $\delta(S_1, 0) = S_2$
  - $\delta(S_1, 1) = S_1$
  - $\delta(S_2, 0) = S_1$
  - $\delta(S_2, 1) = S_2$

状态S_1表示在输入的字符串中有偶数个0，而S_2表示有奇数个0。在输入中1不改变自动机的状态。当读完输入的字符串的时候，状态将显示输入的字符串是否包含偶数个0。

$\mathcal{A}$能识别的语言是$\mathcal{L}(\mathcal{A}) = \{w | \#_0(w) \equiv 0 \ (mod\ 2)\}$。用正则表达式表示为：$(1^*(01^*0)^*)^*$。

封闭性及一些运算

封闭性

确定有限状态自动机的交，并，差，补，连接，替换，同态，逆同态等运算是封闭的，也就是说确定有限状态自动机通过这些运算产生的新的自动机也是确定有限状态自动机。

补运算

$\mathcal{A}=(Q,\Sigma,\delta,s,F)$是一个DFA，那么由补运算产生的新DFA定义为：$\bar{\mathcal{A}}=(Q,\Sigma,\delta,s,Q-F)$。显然只要将$\mathcal{A}$中接受的状态设为不接受的状态，同时把不接受的状态设为接受的状态就得到$\bar{\mathcal{A}}$。补运算的复杂度是：$O(|Q|)$。

交运算和并运算

有两个DFA，$\mathcal{A}_1=(Q_1,\Sigma,\delta_1,s_1,F_1)$和$\mathcal{A}_2=(Q_2,\Sigma,\delta_2,s_2,F_2)$，那么由这两个DFA创造出来的新的自动机定义为：$\mathcal{B}=(Q_1\times Q_2,\Sigma,\delta_{\mathcal{B}},(s_1,s_2),M)$。其中$M\subseteq Q_1\times Q_2$，$(s_1,s_2)$为$\mathcal{B}$的开始状态，$\delta_{\mathcal{B}}$为$\mathcal{B}$的转移函数，且作如下定义：$\forall q_1\in Q_1,\ q_2\in Q_2,\ \sigma\in\Sigma:\delta_{\mathcal{B}}((q_1,q_2),\sigma)=(\delta_1(q_1,\sigma),\delta_2(q_2,\sigma))$。

1. 当$M=F_1\times F_2$时，由上述方法得到的$\mathcal{B}$就是DFA $\mathcal{A}_1$和$\mathcal{A}_2$的交运算，记作：$\mathcal{B}=\mathcal{A}_1\cap\mathcal{A}_2$。也就是说对于读入的字符串w，当且仅当$\mathcal{A}_1$和$\mathcal{A}_2$同时接受w的时候$\mathcal{B}$接受w。
2. 当$M=Q_1\times F_2\bigcup F_1\times Q_2$时，由上述方法得到的$\mathcal{B}$就是DFA $\mathcal{A}_1$和$\mathcal{A}_2$的并运算，记作：$\mathcal{B}=\mathcal{A}_1\cup\mathcal{A}_2$。也就是说对于读入的字符串w，只要$\mathcal{A}_1$或$\mathcal{A}_2$中至少有一个接受w，$\mathcal{B}$就接受w。

交运算和并运算的复杂度都是$O(|Q_1||Q_2||\Sigma|)$。

同态和逆同态运算

一个同态函数$h:\Sigma^*\to\Gamma^*$可以递归的定义为：

- $h(\epsilon)=\epsilon$
- $h(u\sigma)=h(u)h(\sigma)$

于是则有 $h(uv)=h(u)h(v)$。（以上所述中 ϵ为空字符， $u,v\in\Sigma^*,\sigma\in\Sigma$）

1. $\mathcal{L}\subseteq\Sigma^*:h(\mathcal{L}):=\{h(w)\mid w\in\mathcal{L}\}$ ：对于接受语言L的DFA，只要将其中代表 $\delta(q,\sigma)$的边替换成一个序列 $h(\sigma)$并在其中加入不属于原DFA状态的新状态，就产生了接受语言h(L)的DFA。
2. $\mathcal{L}\subseteq\Gamma^*:h^{-1}(\mathcal{L}):=\{w\mid h(w)\in\mathcal{L}\}$ ：定义一个 Q,Σ,s,F都不变的新DFA，并定义新的转移函数为 $\delta'(q,\sigma):=\delta^*(q,h(\sigma))$，则 (Q,Σ,δ',s,F)就是逆同态运算产生的新DFA。

此外替换运算和逆同态运算的方法近似。

最小自动机

等价类自动机

对于一个正则语言，接受该语言的等价类自动机是一个 $(Q, \Sigma, \delta, s, F)$的5-元组。其定义如下：

- Q是等价关系~L的等价类的集合：$[x], x \in \Sigma^*$的集合
- $s = [\epsilon]$
- $F = \{[x] \mid x \in L\}$
- $\delta([x], \sigma) = [x\sigma]$

~L被称为Nerode关系，是Myhill-Nerode定理的基础。简单的来说就是对于任意 $x, y, z \in \Sigma^*$ ，如果$xz \in L \Leftrightarrow yz \in L$，那么x~Ly。

唯一性

对于任意给定的确定有限状态自动机都能找到一个与之计算能力等价的最小确定有限状态自动机，简称最小自动机。该最小自动机中状态的数量等于能识别相同语言的等价类自动机中等价关系的数量，我们可以称最小自动机和等价类自动机“实际上”是相等的，也就是同构。非正式的说法是：对于最小自动机上的任意状态都可以通过一个同构函数变换成等价类自动机上的一个状态。

能识别一个正则语言的等价类自动机是唯一的，因此能识别该语言的最小自动机也是唯一的。

算法

定义一个非等价关系：$N := \{(p, q) \mid p, q \in Q, \exists w \in \Sigma^* : \delta^*(p, w) \in F \leftrightarrow \delta^*(q, w) \notin F\}$，如下步骤可以得到这个集合N：

1. 如果$p \in F,\ q \notin F$ ，就给所有的状态对(p,q)和(q,p)打上标记
2. 重复步骤3，直到所标记的状态对没有变化为止
3. 对于未标记的状态对(p,q)和σ，如果 $(\delta(p, \sigma), \delta(q, \sigma))$被标记过了就把(p,q)也标记上
4. 以上所有标记了的状态对的集合就是非等价关系N

以下是由一个任意DFA转换到一个最小DFA的步骤：

1. 把所有不能从开始状态达到的状态删除
2. 通过上述标记算法计算非等价关系N
3. 一步一步将不属于N的状态对中的两个状态合成一个状态

这样就得到了接受相同语言的最小自动机。复杂度为$O(|Q|^2\ |\Sigma|)$。

引用

- Michael Sipser, *Introduction to the Theory of Computation*. PWS, Boston. 1997. ISBN 0-534-94728-X. Section 1.1: Finite Automata, pp.31–47. Subsection "Decidable Problems Concerning Regular Languages" of section 4.1: Decidable Languages, pp.152–155.4.4 DFA can accept only regular language

参见

- 无环确定有限状态自动机

- 非确定有限状态自动机
- 图灵机
- 读只右移图灵机
