目录

1 介绍

自 1950 年来，有限自动机的就一直在研究当中。简单来说就是找到一个唯一的（直至同构）最小的确定性有限自动机，它能接收与给定的确定性有限自动机相同的语言。解决这一问题的算法应用范围很广，从编译器构造到硬件电路的最小化都有它的身影。有了形式多样的应用程序，不同的表示形式的数量也在增加: 大多数教科书都有自己的变体，而时间复杂度最优的算法 (Hopcroft 的算法) 仍然晦涩难懂。

本文介绍了有限自动机最小化算法的有关分类。如下所示:

- 大多数教材的作者称他们的最小化算法由 Huffman 算法 [Huff54] 和 Moore 算法 [Moor56] 直接推导得到。不幸的是，大多数教材都展示了截然不同的算法 (比如 [AU92], [ASU86], [HU79], [Wood87]), 只有由 Aho 和 Ullman 发表的算法直接源自 [Huff54, Moor56]。
- 虽然大多数算法依赖于计算状态的等价关系，但伴随算法演示的许多解释并未明确提及算法是计算等价关系、它包含的状态划分还是它的补充。
- Comparison of the algorithms is further hindered by the vastly differing styles of presentation — sometimes as imperative programs or as functional programs, but frequently only as a descriptive paragraph. 算法之间的比较进一步受到呈现方式的巨大差异的阻碍。有时作为命令式程序或函数式程序，但通常只作为描述性段落。

A related taxonomy of finite automata construction algorithms appears in [Wats93].

有限自动机构造算法的相关分类在 [WATS93] 中。

All except one of the algorithms rely on determining the set of automaton states which are equivalent. The algorithm that does not make use of equivalent states is discussed in Section 2. In Section 3 the definition and some properties of equivalence of states is given. Algorithms that compute equivalent states are presented in Section 4. The main results of the taxonomy are summarized in the conclusions Section 5. Appendices A and B give the basic definitions required for reading this paper. The definitions related to finite automata are taken from [Wats93]. The minimization algorithm relationships are shown in a ”family” tree in Figure 1.

除了一个算法之外，所有依赖于确定等价的自动机状态的集合。在第 2 节中讨论了不使用等效状态的算法。在第 3 节中给出了状态等价的定义和一些性质。计算等效状态的算法在第 4 节中给出。分类的主要结果总结在结论部分 5 中。附录 A 和 B 给出了阅读本文所需的基本定义。与有限自动机相关的定义取自 [WATS93]。图 1 中的“家庭树”中显示了最小化算法关系。

The principal computation in most minimization algorithms is the determination of equivalent (or inequivalent) states ——thus yielding an equivalence relation on states. In this paper,we consider the following minimization algorithms:

大多数最小化算法的主要计算是确定等价的（或不等价的）状态，从而在状态上产生等价关系。在本文中，我们考虑以下最小化算法：

- Brzozowski’s (possibly nondeterministic) finite automaton minimization algorithm as presented in [Brzo62]. This elegant algorithm (Section 2) was originally invented by Brzozowski, and has since been re-invented without credit to Brzozowski. Given a (possibly

nondeterministic) finite automaton without E-transitions, this algorithm produces the minimal deterministic finite automaton accepting the same language.

Brzozowski（可能是非确定性的）有限自动机最小化算法在 [BRZO62] 中提出。这个优雅的算法（第 2 节）最初是由 Brzozowski 发明的，此后又在没有 Brzozowski 的功劳情况下被重新发明。在没有 ϵ- 跃迁的情况下，给出了一个（可能不确定的）有限自动机，该算法产生最小的确定的有限自动机接受相同的语言。

- Layerwise computation of equivalence as presented in [Wood87, Moor56 Brau88, Urba89]. This algorithm (Algorithm 4. 2) is a straightforward implementation suggested by the approximation sequence arising from the fixed-point definition of equivalence of states.

  分层等价计算等价于 [Wood87, Moor56 Brau88, Urba89] 中提出。算法（算法 4. 2）是由状态等价的定点定义产生的近似序列所建议的直接实现。

- Unordered computation of equivalence This algorithm (Algorithm 4.3, not appearing in the literature) computes the equivalence relation; pairs of states (for consideration of equivalence) are chosen in an arbitrary order.

  该算法（算法 4.3，未出现在文献中）计算等价关系；以任意顺序选择状态对（考虑等价性）。

- Unordered computation of equivalence classes as presented in [ASU86]. This algorithm (Algorithm 4.4)is a modification of the above algorithm computing equivalence of states.

  在 [ASU86] 中给出的等价类的无序计算。该算法（算法 4.4）是上述算法计算状态等价性的一种改正。

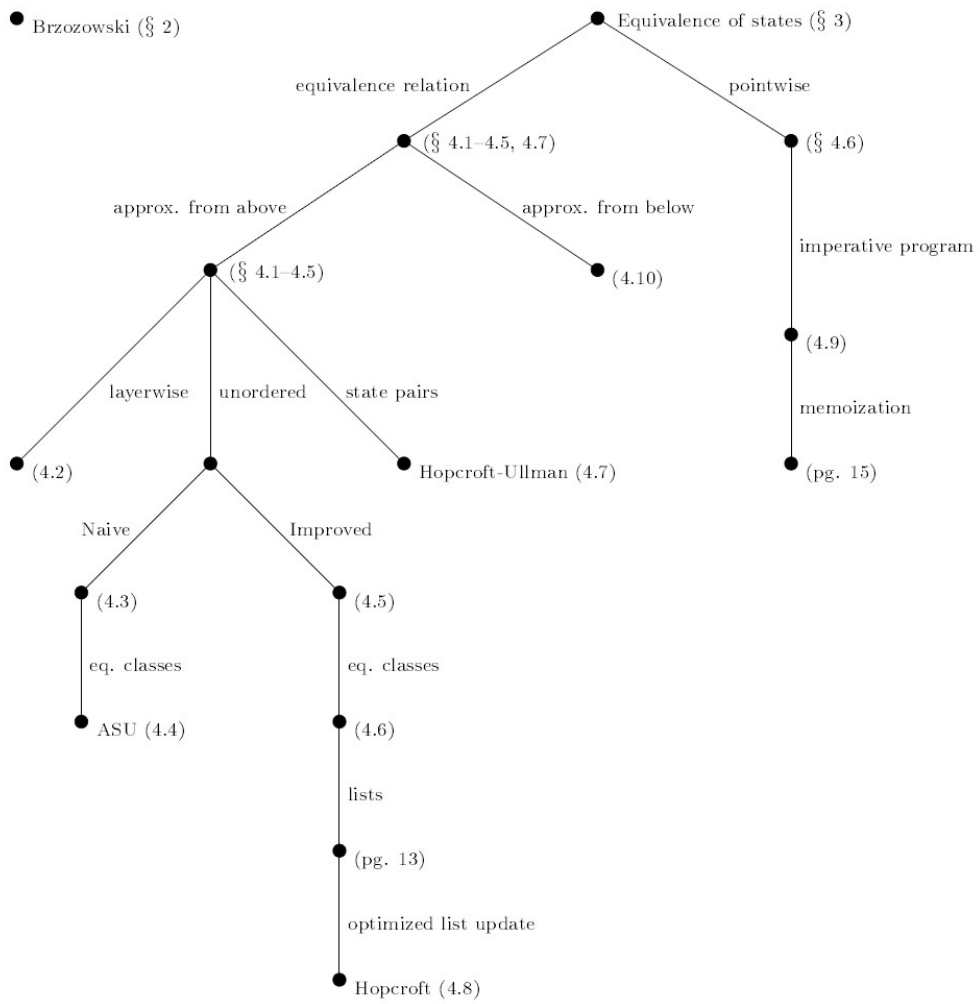

Figure 1: The family trees of finite automata minimization algorithms. Brzozowski's minimization algorithm is unrelated to the others, and appears as a separate(single vertex)tree. Each algorithm presented in this paper appears as a vertex in this tree. For each algorithm that appears explicitly in this paper, the construction number appears in parentheses(indicating where it appears in this paper). For algorithms that do not appear explicitly, a reference to the section or page number is given. Edges denote a refinement of the solution (and therefore explicit relationships between algorithms). They are labeled with the name of the refinement.

图 1：有限自动机最小化算法的关系树。Brzozowski 的最小化算法与其他算法无关，并作为一个单独的（单顶点）树出现。本文提出的每一个算法都作为树的顶点出现。对于本文中明确出现的每个算法，构造数在括号中（标示它在本文中出现的位置）。对于未显式显示的算法，给出了相应的页码。边表示解的细化（因此算法之间的显式关系）。它们被标记为细化的名称。

- Improved unordered computation of equivalence. This algorithm (Algorithm 4.5, not appearing in the literature) also computes the equivalence relation in all arbitrary order. The algorithm is a minor improvement over the other unordered algorithm.

  改进的等价的无序计算。这个算法 (算法 4.5，没有出现在文献中) 也以任意顺序计算等价关系。该算法是对其他无序算法的一个小改进。

- Improved unordered computation of classes. This algorithm (Algorithin 4.6, not appearing in

the literature) is a modification of the above algorithm to compute the equivalence classes of states. This algoritm is used in the derivation of Hopcroft's minimization algorithin.

改进了类的无序计算。该算法（算法 4.6，不在文献中）是上述算法的修改，用来计算等价类的状态。该算法用于 Hopcroft 最小化算法的推导。

- Hopcroft and Ullman's algorithm as presented in [HU79]. This algorithm (Algorithm 4.7) computes the inequivalence (distinguishability) relation. Although it is based upon the algorithus of Huffinan and Moore [Huff54, Moor 56], this algorithm uses some interesting encoding techniques.

  Hopcroft 和 Ullman 算法在 [HU79] 中提出。该算法（算法 4.7）计算不等价（区分性）关系。虽然它是基于 Huffinan 和 Moore [Huff54，MOR 56] 的算法，但该算法使用一些有趣的编码技术。

- Hopcroft's algorithm as presented in [Hopc71, Grie73]. This algorithm (Algorithm 4.8)is the best known algorithm (in terms of running time analysis) for minimization. As the original presentation by Hopcroft is diflicult to understand, the presentation in this paper is based upon the one giveu by Gries.

  HopRofft 的算法在 [Hopc71, Grie73] 提出的。该算法（算法 4.8）是用于最小化的最有名的算法（在运行时间分析方面）。由于 Hopcroft 的原始陈述是难以理解的，本文的介绍基于 Gries 的文章。

- Pointwise computation of equivalence. This algorithm (Algorithm 4.9 not appearing in the literature) computes the equivalence of a given pair of states. It draws upon some nonautomata related techniques, such as: structural equivalence of types and memoization of functional programs.

  等价的点态计算。该算法（算法 4.9，不在文献中）计算给定状态对的等价性。它借鉴了一些非自动机相关的技术，例如：类型的结构等价和函数式程序的记忆化。

- Computation of equivalence from below (with respect to refinement). This algorithm (Algorithm 4.10, not appearing in the literature) computes the equivalence relation from below. Unlike any of the other known algorithms, the intermediate result of this algorithm can be used to construct a smaller (although not minimal) deterministic finite automaton.

  由下面的内容（关于细化）计算等价性。该算法（算法 4.10，不在文献中）计算从下面的等价关系。与任何其他已知算法不同，该算法的中间结果可用于构造较小的（虽然不是最小的）确定性有限自动机。

2 Brzozowski 提出的算法

Most minimization algorithms are applied to a DFA. In the case of a nondeterministic FA, the subset construction is applied first, followed by the minimization algorithm. In this section, we consider the possibility of applying the subset construction (with useless state removal) after an (as yet unknown) algorithm to yield a minimal DFA. We now construct such an algorithm. (The algorithm described in this section can also be used to construct the minimal Complete DFA, by replacing function subsetopt with subset.)

大多数最小化算法应用于确定的有限自动机 (DFA)。对于不确定性有限自动机，首先应用子集构造，然后应用最小化算法。在本节中，我们将考虑在 (未知的) 算法之后应用子集构造 (带无用状态删除) 以生成最小 DFA 的可能性。我们现在构造这样的算法。（在本节中描述的算法也可用于通过用 *subset* 替换函数 *subsetopt* 来构造最小完全 DFA）。

Let $M_0 = (Q_0, V, T_0, \emptyset, S_0, F_0)$ be the $\epsilon - free$ FA.

This algorithm was originally given by Brzozowski in[Brzo62]. The origin of this algorithm was obscured when Jan van de Snepscheut presented the algorithm in his Ph.D thesis [vdSn85]. In this thesis, the algorithm is attributed to a private communication from Prof. Peremans of the Eindhoven University of Technology. Peremans had originally found the algorithm in an article by Mirkin [Mirk65]. Although Mirkin does cite a paper by Brzozowski[Brzo64], it is not clear whether Mirkin's work was influenced by Brzozowki's work on minimization. Jan van de Snepscheut's recent book [vdSn93] describes the algorithm, but provides neither a history nor citations (other than his thesis for) this algorithm.

该算法最初是由 Brzozowski 在 [Brzo62] 中给出。最初在 Jan van de Snepscheut 在他的博士论文 [vdSv885] 中提出该算法时是模糊的。本文中，算法的起因是一个教授的私人通讯。埃因霍芬理工大学的 Pereman, Mirkin [Mirk65] 的文章中找到了该算法。虽然 Mirkin 引用了 Brzozowki [Brzo64] 的论文，但米尔金的作品是否受 Brzozowki 的最小化工作的影响尚不清楚。Jan van de Snepscheut 的新书 [VDSn93] 描述了该算法，但既不提供该算法历史，也不提供该算法的引文（除他的论文外）。

3 状态等价最小化

本节中，我们限定在完全 DFA 的最小化中。This is strictly a notational convenience, as the miniization algorithm can be modified to work for non-complete DFA. 严格地说，这是一种标记的便利，因为可以对缩小算法进行修改，使其适用于不完全 DFA。完全最小化 DFA 比不完全最小化 DFA 多一个状态 (下沉态)，unless the language of the DFA is V^*。把 $M = (Q, V, T, \theta, S, F)$ 设为完全 DFA，这个特殊的 DFA 将会贯穿本节内容。我们也假设所有的 M 的所有状态都是开始可达的。

为了最小化 DFA M，我们计算等价关系 $E \subseteq Q \times Q$, 将其定义为

$(p, q) \in E \equiv (\overrightarrow{\mathcal{L}}(p) = \overrightarrow{\mathcal{L}}(p))$

Since this is an equivalence relation, we are really interested in unordered pairs of states. 由于这是一个等价关系，我们确实对无序状态对有兴趣，使用有序对比使用无需对更方便。

根据等价关系 E ，使用 *merge* 转换来对等价关系进行转换。

转换 *3.1 (*合并状态*)* : 对于任何满足 $H \in E$ 的等价关系 H, 函数 $merge$ 可以用来减少 DFA^2 (当 H 是状态上的恒等关系时，函数 $merge$ 不会减少状态数) 的状态数，函数 $merge$ 定义为：

$$merge((Q, V, T, \theta, \{s\}, F), H) = \text{let} \quad T' = \{([p]h, a, [q]h) : (p, a, q) \in T\}$$
$$\text{in}$$
$$([Q]_H, V, T', \theta, \{[s]_H\}, [F]_H)$$
$$\text{end}$$

$merge$ 的定义依赖于等价类代表的选择。函数 $merge$ 拥有以下性质：

$$\mathcal{L}_{FA}(merge(M, H)) = \mathcal{L}_{FA}(M) \wedge |merge(M, H)| \leq |M| \wedge |merge(M, H)| = \#H$$

and it preserves $Complete, \epsilon - free, Useful, Det, and minimal$; indeed, $merge$ is only defined on $\epsilon - free$ and deterministic FA's.

为了计算关系 E, 需要函数 $\overrightarrow{\mathcal{L}}$ 的一个性质。

性质 **3.2 (**函数 $\overrightarrow{\mathcal{L}}$**)**: 函数 $\overrightarrow{\mathcal{L}}$ 满足

$$\overrightarrow{\mathcal{L}}(p) = (\cup a : a \in V : \{a\} \cdot \overrightarrow{\mathcal{L}}(T(p, a))) \cup (\text{if}(p \in E)\text{then}\{\epsilon\}\text{else}\theta\text{fi})$$

This allow us to give an alternate characterization of equivalence of states. 这允许我们给出状态等价的另一种描述。

定义 **3.3 (**状态的等价**):** 等价关系 E 是最大不动点 (在优化的情况下)

$$(p, q) = E \equiv (p \in F \equiv q \in F) \wedge (\forall a : a \in V : (T(p, a), T(q, a)) \in E)$$

备注 3.4：the greatest fixed point has the least number of equivalence classes of any such fixed point. 最大不动点具有任何此类不动点的等价类数最少。

备注 3.5：定义 3.3 中的任何不动点都可以使用。为了最小化自动机，需要最大不动点。

性质 3.6(近似 E)：我们可以用连续逼近来计算最大不动点，E 的连续逼近如下 ($k \geq 0$):

$$(p, q) \in (E_{k+1} \equiv (p, q) \in E_k \wedge (\forall a : a \in V : (T(p, a), T(q, a)) \in E_k)$$

E_0 定义为：

$$(p,q) \in E_0 \equiv (p \in F \equiv q \in F)$$

E_0 的一个等价定义是 $E_0 = (q \setminus F)^2 \cup F^2$。对于所有的 $K \geq 0$ 有 $E_{k+1} \in E_k$。

备注 3.7：如果 E_K 是一个等价关系，那么 E_{k+1} 也是。E_0 是一个等价关系。

备注 3.8：有一个直观的 E_k 的说明是有用的。当且仅当没有字符串$w : |w| \leq k$ 满足$w \in \overrightarrow{\mathcal{L}}(p) \not\equiv w \in \overrightarrow{\mathcal{L}}(q)$ 一个状态对p, q 也可以说成$k-$等价 (写作$(p,q) \in E_k$)。作为结果，当且仅当

- 两者都是最终态或者都不是最终态；
- 所有的 $a \in V, T(p,a)$和$T(q,a)$ 都是 $(k-1)-$等价 (根据 $\overrightarrow{\mathcal{L}}$ 和 T^* 的定义)；

时 p 和 q 都是 $k-$等价。

备注 3.9：E 的一个重要性质是，它也是包含于（set containment instead of refinement）定义 3.3 中等价性的最大的固定点。作为最大的不动点，E 可以用 $\subseteq -descending$ 关系序列来计算，从 $Q \times Q$ 开始，这样的序列不必只包含等价关系。在这样的近似序列中可能有比在上面给出的 E_k 序列有更多的步骤。幸运的是，每个这样的步骤通常都比从 E_k 计算 E_{k+1} 更容易计算。较容易计算这些（但较长）序列的一些算法在第 4.2-4.5 和第 4.7 节中给出。

所有先前已知的算法都是通过上面的逐次逼近法（相对于 $\subseteq$）计算 E。第 4.7 节中的新算法通过以下逐次逼近来计算 E。在这一节中，解释了这一点的实际重要性。

3.1 可分辨性

通过首次计算它的 complement $D = \neg E$ 来计算 E 是有可能的。关系 D(也叫做状态关系的可分辨性) 定义为

$$(p,q) \in D \equiv (\overrightarrow{\mathcal{L}}(p) \neq \overrightarrow{\mathcal{L}}(q))$$

定义 3.10(状态的可分辨性)：D 是一个方程中的最小 (under $\in$, set containment) 不动点

$$(p,q) \in D \equiv (p \in F \not\equiv q \in F) \wedge (\exists a : a \in V : (T(p,a)) \in D)$$

性质 3.11 (逼近 D)：随等价关系 E，关系 D 可以通过连续逼近来计算 ($k \geq 0$)

$$(p,q) \in D_{k+1} \equiv (p,q) \in D_k \wedge (\exists a : a \in V : (T(p,a), T(q,a)) \in D_k)$$

有 $D_0 = \neg E_0 = ((Q \setminus F) \times F) \cup (F \times (Q \setminus F))$，对于所有的 $k \geq 0$, 有 $D_k = \neg E_k$，同时 $D_{k+1} \subseteq D_k$。

备注 3.12：对于 E_k，有一个直观的 D_k 的说明是有用的。当且仅当没有字符串$w : |w| \leq k$ 满足$w \in \overrightarrow{\mathcal{L}}(p) \not\equiv w \in \overrightarrow{\mathcal{L}}(q)$ 一个状态对p, q 也可以说成$k-$等价 (写作$(p,q) \in E_k$)。作为结果，当且仅当

- 其中一个是最终态而另外一个不是最终态；
- 存在 $a \in V$ such that $T(p,a)$ 和 $T(q,a)$ 是在 (k-1)-distinguished。

时 p 和 q 是 k-distinguished （一些作者也把它叫做 k-distinguishable）。

3.2 近似步骤数的上界

我们可以很容易的把一个近似步骤数的上界放进 E 的计算当中。

设 E_j 为定义 E 的方程的最大不动点，可以得到以下逼近步骤（where I_Q 是状态上的恒等关系）：

$$E_0 \subset E_1 \subset \cdots \subset E_j \in I_Q$$

近似序列中部分等价关系的指数是已知的：$\sharp I_Q = |Q|$且$\sharp E_0 \leq 2$，可以推导出：

$$\sharp E_0 < \sharp E_1 < \cdots < \sharp E_j \leq \sharp I_Q = |Q|$$

$\sharp E_0 = 0$ 时，E_0 是最大不动点。$\sharp E_1 = 1$ 时，要么所有状态都是终态，要么所有状态都不是终态。两种情况下 E_0 都是最大不动点。$\sharp E_0 = 2$ 时，$i+2 \leq \sharp E_i$，由 $j+2 \leq \sharp E_j \leq \sharp I_Q = |Q|$ 可得 $j \leq |Q|-2$ 。这给出了计算（从 E_0 开始）最大不动点 E_j 的步骤 $(|Q|-2)$ 的上界，最大值 0（使用性质 3.6 中的逼近序列）。

上界为 $E = E_{(|Q|-2)max0}$。正如之后我们会见到的那样，它可以为算法带来效率上的提升。这个结果同样在 Wood [Wood87, Lemma 2.4.1] 中有所记录。This upperbound also holds for computing D and $[Q]_E$ by approximation。

3.3 描述 E 的等价类

计算 $[Q]_E$: E 的等价类集合同样是可行的。In order to partition $[Q]_E$ 我们由定义 3.3 出发，把 E 的最大等价关系的描述为：

$$\begin{aligned}
&(\forall p,q:(p,q)\in E:(p\in F\equiv q\in F)\wedge(\forall a:a\in V:(T(p,a),T(q,a))\in E))\\
\equiv\ &\{\text{ definition of membership in E; more a to outer quantification }\}\\
&(\forall p,q,a:(p,q)\in E\wedge a\in V:(p\in F\equiv q\in F)\wedge[T(p,a)]_E=[T(q,a)_E])\\
\equiv\ &\{\text{ Introduce equivalence classes } Q_0,Q_1 \text{ explicity}\}\\
&(\forall Q_0,Q_1,a:Q_0\in[Q]_E\wedge Q_1\in[Q]_E\wedge a\in V:\\
&(\forall p,q:p\in Q_0\wedge q\in Q_0:(p\in F\equiv q\in F)\wedge T(p,a)\in Q_1\equiv T(q,a)\in Q_1))
\end{aligned}$$

定义 3.13（函数 $Splittable$）: 为了使其更简洁，我们定义

$$Splittable(Q_0,Q_1,a)\equiv(\exists p,q:p\in Q_0\wedge q\in Q_0:(T(p,a)\in Q_1\not\equiv T(q,a)\in Q_1))$$

使用 $Splittable, [Q]_E$is the largest partition (under $\sqsubseteq$) such that $[Q]_E \sqsubseteq [Q]_{E_0}$ and

$$(\forall Q_0,Q_1,a:Q_0\in[Q]_E\wedge Q_1\in[Q]_E\wedge a\in V:\neg Splittable(Q_0,Q_1,a))$$

可以用在 $[Q]_E$ 的计算中。

4 计算 $E, D, [Q]_E$ 的算法

本节中，对计算 $E, D, [Q]_E$ 的算法进行叙述。一些算法以通用形式发表：计算 D 和 E。由于只需要 D 和 E 之中的一个（不是两个都需要），在实际使用中，可以更改通用算法来只计算其中一个。

4.1 通过分层逼近来计算 D 和 E

根据 E_k，E_{k+1} 的定义（D 也适用）很自然的引出下面计算 D 和 E 的算法（其中 k 是一个 ghost 变量，仅用于指定不变量）。

算法 **4.1**

$G, H = D_0, E_0$;
$G_{old}, H_{old}, k = \emptyset, Q \times Q, 0$;
{invariant: $G = D_k \wedge H = E_k$}
do $G \neq G_{old} \longrightarrow$
$\{G \neq G_{old} \wedge H \neq H_{old}\}$
$G_{old}, H_{old} := G, H$;
$G := (\cup p, q : (p, q) \in G_{old} \wedge (\exists a : a \in V : (T(p, a), T(q, a)) \in G_{old}) : \{(p, q)\})$;
$G := (\cup p, q : (p, q) \in H_{old} \wedge (\forall a : a \in V : (T(p, a), T(q, a)) \in H_{old}) : \{(p, q)\})$
$\{G = \neg H\}$
$k := k + 1$
od $\{G = D \wedge H = E\}$

This algorithm is said to compute D and E layerwise, since it computes the sequences D_k and E_k.The update of G and H in the repetition can be made with another repetition as shown in the program now following.

算法 **4.2**

$G, H = D_0, E_0$;
$G_{old}, H_{old}, k = \emptyset, Q \times Q, 0$;
{invariant: $G = D_k \wedge H = E_k$}
do $G \neq G_{old} \longrightarrow$
$\{G \neq G_{old} \wedge H \neq H_{old}\}$
$G_{old}, H_{old} := G, H$;
for $(p, q) : (p, q) \in H_{old}$ do
if $(\exists a : a \in V : (T(p, a), T(q, a)) \in G_{old}) \longrightarrow G, H := G \cup (p, q), H \setminus (p, q)$
$|(\exists a : a \in V : (T(p, a), T(q, a)) \in H_{old}) \longrightarrow$ skip
fi
rof
$\{G = \neg H\}$
$k := k + 1$
od $\{G = D \wedge H = E\}$

此算法可以分为两部分：一部分只计算 D，另外一部分只计算 E。只计算 E 的算法本质上是由 Wood 在 [Wood87,pg.132] 发表的。据 Wood 称，此算法建立在 Moore [Moor56] 的基础之上。他的时间复杂度为 $O(|Q|^3)$ 。在 [Brau99] 中，Brauer 使用了一些编码技术提供了此

算法的时间复杂度为 $O(|O|^2)$ 的版本，而后 Urbanek 在 [Urba89] 中提供了 Brauer 的控件优化版本。这里没有给出任何一个它们的变体。仅计算 D 的算法本文中没有提及。

只要稍稍多做一点工作，这个算法就可更改用来计算 $[Q]_E$。

4.2 通过无序逼近来计算 D，E 和 $[Q]_E$

我们可以用任意顺序的状态对来计算 E，而不是计算每一个 E_k（按层计算 E）(如备注 3.9 所示)。使用下面的算法可以做到（也可以用来计算 D）：

算法 **4.3**

$G, H = D_0, E_0$;
{invariant: $G = \neg H \wedge G \subseteq D$}
do $(\exists p, q, a : a \in V \wedge (p, q) \in H : (T(p, a), T(q, a)) \in G) \longrightarrow$
　　let $p, q : (p, q) \in H \wedge (\exists a : a \in V : (T(p, a), T(q, a)) \in G)$
　　$\{(p, q) \in D\}$
　　$G, H := G \cup (p, q), H \setminus (p, q)$
od$\{G = D \wedge H = E\}$

此算法可以分解为一个只计算 D 和一个只计算 E 的算法。在每个迭代步骤的最后，H 可能不是一个等价状态（也就是 $H \neq H^*$）——详见备注 3.9。可以通过在 **od** 前面添加一个 assignment 来对这个算法稍作修改:

$$H := (\mathbf{MAX}_{\subseteq} J : J \subseteq H \wedge J = J^* : J); G := \neg H$$

assignment 的添加使得算法可以计算细化序列 E_k（详见备注 3.9）. 如果使用计算量化 **MAX** 的简便方法，那么这个 assignment 可以提升这个算法的时间复杂度。本文中未提及此算法。

当我们把上面的这个算法转化来计算 $[Q]_E$，最终的算法如下，由 Aho, Sethi 和 Ullman 在 [ASU86,Alg.3.6] 中提出：

算法 **4.4**

$P : [Q]_{E_0}$;
{incariant:$[Q]_E \sqsubseteq p \sqsubseteq [Q]_{E_0}$}
do $(\exists Q_0, Q_1, a : Q_0 \in P \wedge Q_1 \in P \wedge a \in V : Splittable(Q_0, Q_1, a)) \longrightarrow$
　　let $Q_0, Q_1, a : Splittable(Q_0, Q_1, a)$;
　　$Q'_0 := \{p : p \in Q_0 \wedge T(p, a) \in Q_1\}$;
　　$\{\neg Splittable(Q_0 \setminus Q'_0, Q_1, a) \wedge \neg Splittable(Q'_0, Q_1, a)\}$
　　$P := P \setminus \{Q_0\} \cup \{Q_0 \setminus Q'_0, Q'_0\}$
od
$\{(\forall Q_0, Q_1, a : Q_0 \in P \wedge Q_1 \in P \wedge a \in V : \neg Splittable(Q_0, Q_1, a))\}$
$\{P = [Q]_E\}$

此算法时间复杂度为 $O(|Q|^2)$。

4.3 通过无序逼近更高效的计算 D 和 E

我们提出另外一种以任意顺序状态对考虑的算法。该算法（也可以计算 D）由两个嵌套循环组成：

算法 **4.5**

$G, H = D_0, E_0$;
$\{invariant : G = \neg H \wedge G \subseteq D\}$
do $(\exists p, q, a : a \in V \wedge (p, q) \in H : (T(p, a), T(q, a)) \in G) \longrightarrow$
  let $p, a : p \in Q \wedge a \in V \wedge (\exists q : (p, q) \in H : (T(p, a), T(q, a)) \in G)$;
  for $q : (q, p) \in H \wedge (T(p, a), T(q, a)) \in G$ **do**
    $G, H := G \cup \{(p, q)\}, H \setminus \{(p, q)\}$
  rof
od $\{G = D \wedge H = E\}$
