[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/06 (2006.01)

[12] 发明专利申请公布说明书

[21] 申请号 200780034213.4

[43] 公开日 2009 年 8 月 26 日

[11] 公开号 CN 101517533A

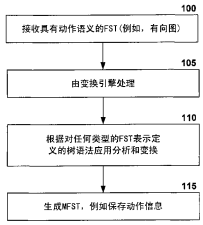
[22] 申请日 2007.9.14
[21] 申请号 200780034213.4
[30] 优先权
[32] 2006. 9.15 [33] US [31] 11/532,349
[86] 国际申请 PCT/US2007/078522 2007.9.14
[87] 国际公布 WO2008/034075 英 2008.3.20
[85] 进入国家阶段日期 2009.3.16
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 S·E·卢科 D·E·兰沃西
G·M·德拉 - 利贝拉
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 陈 斌
权利要求书 3 页 说明书 36 页 附图 25 页

[54] 发明名称

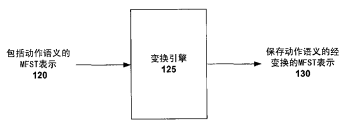
模块化有限状态变换机的变换

[57] 摘要

提供 Q 框架，简称为 QFX，用于以跨一组不同的 FST 表示类型实现对在其表示中支持动作信息的 FST 的动作语义的保存的通用方式执行高效树变换。 QFX 也允许在执行树变换的同时保存有序和无序嵌套信息、支持将非确定性数据结构变换成确定性数据结构，并允许对含有动作语义的机器进行交运算。



A



B

1. 一种用于在计算系统中将至少一个树数据结构变换成至少一个模块化有限状态变换机（MFST）的方法，包括：

接收 100 表示包括用于定义动作信息的动作语义的至少一个有限状态机（FSM）的至少一个树数据结构 140a；以及

对由所述至少一个树数据结构 140a 表示的任何类型的 FSM 模型，将所述至少一个树数据结构变换 120 成至少一个 MFST 190 同时在所述至少一个 MFST 190 中保存所述至少一个树数据结构的所述动作信息。

2. 如权利要求 1 所述的方法，其特征在于，所述接收 100 包括接收表示包括用于定义动作信息的动作语义的至少一个有限状态变换机（FST）的至少一个树数据结构 140a。

3. 如权利要求 1 所述的方法，其特征在于，所述变换 120 包括采用由变换引擎 180 实现的可扩展预定义转换语法 170 变换所述至少一个树数据结构 140a，所述变换不考虑由所述至少一个树结构 140 表示的 FSM 模型的类型而在所述至少一个 MFST 中保存所述动作信息。

4. 如权利要求 1 所述的方法，其特征在于，所述变换 120 包括用预定义的转换语法 170 变换所述至少一个树数据结构 140a，所述转换语法 170 不考虑所述至少一个树结构 140a 所表示的 FSM 模型的类型而保存所述至少一个树数据结构 140a 的有序和无序嵌套信息两者。

5. 如权利要求 1 所述的方法，其特征在于，所述接收 100 包括接收表示包括用于定义动作信息的动作语义的所述至少一个 FSM 的至少一个有向图数据结构。

6. 如权利要求 6 所述的方法，其特征在于，所述接收 100 包括接收表示包括用于定义动作信息的动作语义的所述至少一个 FSM 的至少一个可扩展标记语言（XML）文档。

7. 如权利要求 1 所述的方法，其特征在于，所述变换 120 包括对至少两个树数据结构进行交运算同时在所述至少一个 MFST 190 中保存所述至少一个树数据结构 140a 的所述动作信息。

8. 如权利要求 1 所述的方法，其特征在于，所述接收 100 包括接收至少一个非确定性树数据结构，且所述变换 120 包括将所述至少一个非确定性数据结构变换成确定性的 MFST 190。

9. 一种包括用于执行如权利要求 1 所述的方法的计算机可执行指令的计算机可读介质。

10. 一种用于将至少一个树数据结构变换成至少一个模块化有限状态变换机（MFST）的计算系统，包括：

表示至少一个有限状态机（FSM）的至少一个树数据结构 140a，所述至少一个树数据结构 140a 是用任何类型的 FSM 存储模型表示的；

用于将所述至少一个树数据结构 140a 变换成至少一个 MFST 190，包括匹配包含在所述至少一个树数据结构 140a 中的有序和无序嵌套信息在内的变换引擎组件 180。

11. 如权利要求 10 所述的计算系统，其特征在于，所述至少一个树数据结构 140a 是表示至少一个有限状态变换机（FST）的至少一个有向图。

12. 如权利要求 10 所述的计算系统，其特征在于，所述变换引擎组件 180 参考预定义转换语法 170 将所述至少一个树数据结构变换成所述至少一个 MFST 190，其中所述转换语法在所述至少一个 MFST 190 中保存所述有序和无序嵌套信息两者。

13. 如权利要求 10 所述的计算系统，其特征在于，所述至少一个树数据结构 140a 包括至少一个非确定性树数据结构，且所述变换引擎组件 180 将所述至少一个非确定性数据结构变换成确定性的 MFST 190。

14. 一种用于在计算系统中将有向图数据结构变换成模块化有限状态变换机（MFST）的变换框架，包括：

用于在计算系统中存储用于表示有限状态机（FSM）的各不相同类型的多个有向图数据结构 140a 的装置；以及

变换机 150，基于预定义树语法 170 分析所述多个有向图数据结构 140a，并将所述多个有向图数据结构 140a 变换成至少一个 MFST 190 同时保存所述多个有向图数据结构 140a 的动作语义信息。

15. 如权利要求 14 所述的变换框架，其特征在于，所述变换机 150 对

由所述多个有向图数据结构表示的有限状态机（FSM）执行至少一个控制流程分析算法，所述算法跨任何类型的有向图表示保存所述多个有向图数据结构的所述动作语义信息。

16. 如权利要求 14 所述的变换框架，其特征在于，所述变换机 150 包括结合变换所述多个有向图数据结构 140a 而执行特定绑定处理的变换引擎 180，其中所述特定绑定处理包括在用于匹配所述多个有向图数据结构 140a 的模式的模式匹配进程期间对所述动作语义信息执行绑定。

17. 如权利要求 14 所述的变换框架，其特征在于，所述变换机 150 内联如由所述多个有向图数据结构 140a 中的一有向图数据结构所定义的至少一个变换机定义，同时保存所述有序和无序语义信息。

18. 如权利要求 14 所述的变换框架，其特征在于，所述变换机 150 将所述多个有向图数据结构 140a 的变量绑定编译成激活记录中的槽。

19. 如权利要求 14 所述的变换框架，其特征在于，所述变换机 150 执行对由所述多个有向图数据结构 140a 表示的所述有限状态机（FSM）的寄存器分析。

20. 如权利要求 14 所述的变换框架，其特征在于，还包括：

用于接收所述多个有向图数据结构 140a 中的一有向图数据结构的接口 I1，它根据所述预定义树语法定义有向图数据结构。

模块化有限状态变换机的变换

技术领域

本发明一般针对诸如有限状态变换机等有限状态自动机，尤其涉及其变换。

背景

作为一般背景，也被称为有限状态自动机（FSA）的有限状态机（FSM）是由状态、转移和动作组成的系统的行为的模型。FSM 或 FSA 可使用状态图、状态转移图、状态表、标记有向图、树等按照各种概念方式表示，它们保存构成该机器即自动机的状态、转移和动作的关系。存在不同种类的 FSM 或 FSA 及其等效方式，以及其中某些的各种已知转换。

目前，存在某些图变换机和树变换机，它们实现例如对某些类型的有向图数据结构变换由 FST 建模的一组系统，以形成新的一个或一组 FST，例如新的有向图数据结构的某些方面，然而，这样的系统由于各种原因而受限。例如，可扩展样式表语言变换（XSLT）是用于 XML 文档变换的基于 XML 的语言。当用于变换其他 XML 文档时，该文档不被改变；相反，基于现有的 XML 文档的内容创建新文档。新文档可由处理器以标准 XML 句法或按照诸如 HTML 或纯文本等另一格式串行化（输出）。XSLT 通常用于在不同 XML 模式之间转换数据，或者将 XML 数据转换成网页或 PDF 文档。

实质上，XSLT 允许创建描述其他 XML 文档的变换的 XML 文档，其又可被转换成不同的格式。然而，XSLT 无法考虑到可用 FST 表示的动作语义。在这一方面，动作可能是“当处于预定义状态时，一旦识别某一信息，例如名字，即执行某一动作”。然而，就可为 FST 定义这样的任意动作而言，XSLT 无法处理作为其变换能力一部分的对这样的任意动作的调用。此外，XSLT 无法在机器上执行补、交和并运算的全部。

而且，诸如 XSLT 等现有变换也无法匹配和构成具有诸如有序和无序嵌套（例如，被表示为树结构）的有序和无序分层信息两者的 FSM。尽管，在某些

环境中，存在当变换树时能够处理仅保存有序信息的某些系统，且存在当变换时能够处理仅保存无序信息的某些系统，但是 XSLT 或者任何已知的系统都不包括能跨有向图或树结构保存有序和无序嵌套两者的变换能力。现有技术的有限状态变换机的变换中的这些和其他缺陷在以下更详细地阐述的本发明的各种示例性非限制性实施例的描述后将变得显而易见。

概述

考虑现有技术的上述缺陷，本发明提供用于对 FST 执行高效树变换，例如交、并、补等的一般框架，该框架跨一组不同类型的表示为在其表示中支持动作信息的 FST 跨变换运算保存动作语义。用于对 FST 执行高效树变换的框架也能够在执行树变换的同时保存有序和无序的嵌套信息，且能够支持非确定性数据结构到确定性数据结构的变换。

在一个实施例中，一种方法用于在计算机系统中将指定树结构的数据结构变换成模块化有限状态变换机（MFST）。该方法包括接收指定表示包括用于定义动作信息的动作语义的有限状态变换机（FST）的树结构的数据结构。对由数据结构表示的任何类型的有限状态机（FSM）模型，该方法包括将数据结构变换成 MFST 同时在 MFST 中保存数据结构的动作信息的能力，其中变换包括对数据结构执行任何交、并和补运算或可归约成任何交、并和补运算的任何运算。在本发明的另一方面中，可根据结果 FSM 判定得到的 FSM 是否接受非空输入。

此处提供了简化概述以帮助对以下更详细的描述和附图中的示例性、非限定性实施例的各方面的基本或大体的理解。然而，本概述并不旨在作为详尽的或穷尽的概观。本概述的唯一目的是以简化的形式来介绍与本发明的各种示例性、非限定性的实施例相关的一些高层概念，作为以下更为详细的描述的序言。

附图简述

将参考附图进一步描述本发明的用于变换模块化有限状态变换机的树语法的技术及其相关联的过程，附图中：

图 1A 示出了显示采用由本发明提供的一般变换框架变换 FST 以保存动作语义的示例性过程的示例性、非限定性流程图；

图 1B 示出了显示采用由本发明提供的一般变换框架变换 FST 以保存动作

语义的示例性过程的示例性、非限定性框图；

图 1C 示出了显示用于根据本发明变换 FST 的示例性框架和变换引擎的示例性、非限定性框图；

图 2A 示出了结合本发明的变换引擎提供的示例性接口；

图 2B 和 2C 分别示出了根据本发明的实施例提供的 PushEnvironment（推入环境）和 PopEnvironment（弹出环境）方法的示例性操作。

图 2D 示出了结合本发明的变换引擎提供的另一示例性接口；

图 2E 列出了如在由本发明提供的变换框架中实现的示例性、非限定性∈闭包算法；

图 2F 示出了如由本发明提供的变换框架中实现的容纳动作的子集构造技术的修改；

图 3A 示出了根据本发明定义的转换范例的增强集的动作正则表达式（REGEX）范例，即 α *T* 范例的示例性转换；

图 3B 示出了根据本发明定义的转换范例的增强集的绑定嵌套范例，即 *x* : [*T*]范例的示例性转换；

图 3C 示出了根据本发明定义的转换范例的增强集的绑定嵌套重复范例，即 $x : [T]^{*|+|?}$范例的示例性转换；

图 3D 示出了根据本发明定义的转换范例的增强集的 *x* : (*T*)范例的示例性转换；

图 3E 示出了根据本发明定义的转换范例的增强集的 *x* : *T Ref* 范例，即使用调用的扩充的示例性转换；

图 3F 示出了根据本发明定义的转换范例的增强集的 $x : T\ Ref^{*|+|?}$ 范例，即使用调用的扩充的示例性转换；

图 4A 和 4B 分别在概念上示出有序（或列表）模式和无序（或集合）模式，结合这些模式本发明的 QFX 用于跨 FST 的变换保存有序和无序嵌套信息两者；

图 5A 示出了显示采用由本发明提供的一般变换框架变换 FST 以保存有序和/或无序信息的示例性过程的示例性、非限定性流程图；

图 5B 示出了显示采用由本发明提供的一般变换框架变换 FST 以保存有序

和/或无序信息的示例性过程的示例性、非限定性框图；

图 5C 到 5L 示出了根据本发明在存在有序或标记信息的情况下对状态机的变换的示例性、非限定性的各方面；

图 6A 示出了根据本发明的 QFX 的示例性有限状态变换机；

图 6B 到 6C 分别示出了表示示例性有序嵌套和无序嵌套的树数据结构，结合这些嵌套本发明的 QFX 用于保存有序和无序嵌套信息两者；

图 7A 示出了根据本发明将 FSM 示例性建模成状态转移表作为示例性状态机数据结构；

图 7B 示出了根据本发明将 FSM 示例性建模成状态表作为示例性状态机数据结构；

图 8A 示出了将 FSM 示例性建模成状态转移图；

图 8B 示出了根据本发明将 Moore 模型示例表示成示例性类型的状态机数据结构的变换机；

图 8C 示出了根据本发明将 Mealy 模型示例表示成示例性类型的状态机数据结构的变换机；

图 9 是表示其中可实现本发明的示例性、非限定性网络化环境的框图；

图 10 是表示其中可实现本发明的示例性、非限定性计算系统或操作环境的框图。

详细描述

如背景中所述，用于变换例如往往在计算存储器中表示的标记有向图等 FST 和树数据结构的当前框架，对于在跨所有类型的 FST 变换时保存某些类型的信息而言不够通用。例如，当前框架无法跨许多类型的 FST 保存某些 FST 的动作语义或由 FST 表示的有序和无序嵌套信息两者。

从而，本发明提供此处被称为 Q 框架（即 QFX）的一般框架，用于执行实现对在其表示中支持动作信息的 FST 保存动作语义的高效树变换。用于执行本发明的高效树变换的 Q 框架也能够在执行其高效树变换的同时保存有序和无序嵌套信息。如根据以下描述将理解地，QFX 也允许各种其他新颖的方面，涉及所定义树语法和相应转换的变换能力、变换机图、变换引擎和变换引擎执行的操作、转换范例以及包括变量重命名的确定化（determinization），如将在

以下更详细描述。在本发明的一个实施例中，本发明对树表示或树表示的集合启用至少交、并、补和空测试变换。

作为进一步的介绍，模块化有限状态变换机（MFST）是实现变换的状态机。具体地，MFST 是扩充了被称为“动作”的程序片段的 MFA。常见的动作包括根据存储在环境中的其他对象来构造对象、绑定环境内的变量以及在模式匹配期间执行谓词。

就此方面，在高层，本发明的 Q 框架（QFX）允许程序员定义具有动作的正则树语法并将这些语法转换成 MFST。QFX 支持使用任何域专用语言（DSL）来定义动作。此外，QFX 提供对变量绑定动作的特殊支持，产生被称为属性化树语法的一种树语法。本发明的以下示例性、非限定性实现描述 QFX 如何能够编译和执行属性化树语法。

就此方面，本发明的编译方法因多个原因而是有利的。该编译方法经由优化和其他高性能方法提供对高性能变换的支持。该编译技术也允许对运行时组件的简化，使得可容易地产生简单变换机。如将在以下示例性、非限定性详细描述，此处所述的编译技术也提供一组清楚的语义。就此方面，本发明的编译方法使地程序员能够容易地推理树语法的语义，包括组合（composition）和侧放作用（side effect）。

一般而言，如图 1A 的流程图所示，在 100 本发明在计算系统中启用用于接收具有动作语义信息的各种类型的有限状态变换机（FST）数据结构的一般框架，然后在 105 将框架的变换引擎应用于 FST。然后，在 110，根据任何类型的 FST 表示的预定义树语法进行 FST 的分析和变换，如将在以下更详细描述的。结果，在 115，FST 被变换以生成 MFST，这保存 FST 的动作信息等，例如在对两个或多个 FST 执行任何交、补和并运算的同时保存有序和无序嵌套信息，并允许将非确定性 FST 变换成确定性结果。

图 1B 是大体对应于示出本发明的树语法所允许的一方面的以上过程的框图。变换框架 125 接收包括动作语义的一个或多个 MFST 120，根据任何交、并和补变换运算（和/或其他布尔运算符，带有组合）变换 MFST，并将动作语义信息保存在输出 130 中。

图 1C 示出了显示用于根据本发明变换 FST 的示例性框架和变换引擎的示

例性、非限定性框图。如图所示，在计算系统中可在各个存储元件 140a 到 140n 中找到各种有限状态机表示，在计算环境中具有用于与根据本发明提供的 Q 框架 150 交互的处理器 P。一般而言，Q 框架 150 经由用于变换和生成 MFST 190 的变换引擎 180 接收有限状态机表示 140a 到 140n 的子集。变换参考属性化树语法 170 对有限状态机表示 140a 到 140n 的子集执行分析 160，以便确定如何最佳地将有限状态机表示 140a 到 140n 的子集变换到 MFST 190。也在计算环境中设置接口 I1，用于接收和定义基于并用于预定义的树语法 170 的有向图数据结构。

因此，在各个非限定性实施例中，本发明允许在计算系统中将树数据结构变换成模块化有限状态变换机（MFST）。表示 FSM（例如 FST）的树数据结构包括定义关于通过变换过程保存的 FSM 的动作信息的动作语义。在一个实施例中，对由树数据结构表示的任何类型的 FSM 模型，例如有向图数据结构、XML 文档等，本发明将树数据结构变换成 MFST 并同时保存动作信息。可扩展、预定义转换语法由无论由树数据结构表示的 FSM 模型的类型是什么都保存动作信息、保存树数据结构的有序和无序嵌套信息两者以及将非确定性数据结构变换成确定性 MFST 的变换引擎实现。有利地，本发明可处理 FSM 类型的补、交和并作为变换过程的一部分。

在其它实施例中，本发明包括 Q 框架（QFX），它包括用于执行有向图到模块化有限状态变换机（MFST）的变换的组件和软件，包括在计算系统中分析任何类型的有向图数据结构并基于分析将有向图数据结构变换成 MFST 同时保存有向图数据结构的动作语义的能力。QFX 以跨任何类型的有向图表示保存有向图数据结构的动作语义信息的方式对由有向图数据结构表示的有限状态机（FSM）执行控制流程分析算法。

此外，QFX 包括结合变换有向图数据结构执行专用的绑定处理的变换引擎。专用绑定处理包括在模式匹配期间对动作语义信息执行绑定。而且，变换包括内联如由向图数据结构中的一有向图数据结构所定义的至少一个变换机定义的能力。而且，变换包括将有向图数据结构的变量绑定编译到现有激活记录中的槽，还包括对由有向图数据结构表示的有限状态机（FSM）的寄存器分析。本发明定义的、实现本发明的以上效果和优点的语法可经由诸如 C#接口

等描述包括动作语义信息的有向图数据结构的软件接口实现。

此外，在各个实施例中，本发明实现采用变量绑定的词汇分析。而且，QFX 是可扩展的，提供与不同的变换供应商的高效互操作性。另外，本发明实现对编译和执行性能之间的折衷的控制，即实现可在不采用昂贵编译的情形解释模式以及编译特定模式以供重复使用。最后，运行时组件可通过提供可变粒度的继续点来参与 QFX 调度规程，从而允许使用同一机制提供单个或多个结果。

以下表 I 中简化的树语法句法是可结合根据本发明的编译过程使用的示例性、非限定性模型。

树语法 → 定义+

定义 → 正则表达式名 '=' 选择

选择 → 规则('|' 规则)* (或)

规则 → 动作?规则条款+ (动作正则表达式)

规则条款 → 嵌套正则表达式 动作? (正则表达式动作)

嵌套正则表达式 → 基本符号 (基本)

| 变量绑定?通配符 (通配符)

| 引用 (引用)

| 嵌套正则表达式 嵌套正则表达式 (序列)

| 变量绑定?'[' 嵌套正则表达式 ']' (绑定嵌套)

| 变量绑定?'(' 嵌套正则表达式 ')' (绑定组)

| 嵌套正则表达式 ('+' | '*' | '?') (重复)

| 嵌套正则表达式 '与' 嵌套正则表达式 (与)

引用 → 变量绑定?正则表达式名 (绑定引用)

变量绑定→ 变量 ':'

表 I-简化树语法句法

在以上语法中，可应用的示例性运算法优先顺序是重复>绑定>序列>与>或。在以下描述中，*x* 用于表示变量；*b* 用于表示基本符号（类型或基本类型的值）；*T* 用于表示 MFST；*α* 用于表示动作；而 *T Ref* 用于表示对 *T* 的引用。

参考正则表达式转换成NFA的现有技术并参考NFA的确定化，可例如在Addison-Wesley 出版社 1986 年出版的 Aho 等人的“Compilers: Principles, Techniques and Tools（编译器：原理、技术和工具）”或Addison-Wesley出版社2000年出版的Hopcroft等人的“Introduction to Automata Theory, Language and Computation（自动机理论、语言和计算的介绍）”中找到这样的转换技术的概观。作为现有技术的改进，根据QFX，经由本发明的技术定义的正则树语法可用对MFST的动作扩充。

而且，在Aho等人的文章中讨论的转换支持根据以上语法覆盖以下范例：或、与、基本、序列、重复和引用；然而，为了根据本发明处理（1）动作、（2）嵌套以及（3）变量绑定，对在下表II中示出的以下附加范例需要转换：

$\alpha\, T$ (动作正则表达式)

$T\, \alpha$ (正则表达式动作)

$x : [T]$ (绑定嵌套)

$x : [T]^{*|+|?}$ (绑定嵌套重复)

$x : (T)$ (绑定组)

$x : (T)^{*|+|?}$ (绑定组重复)

$x : T\, Ref$ (绑定引用)

$x : T\, \mathrm{Ref}^{*|+|?}$ (绑定引用重复)

表 II-由 QFX 处理的转换范例的增强集

动作正则表达式和正则表达式动作范例处理具有一般动作的树语法的编译。绑定引用、绑定组、绑定嵌套、绑定引用重复、绑定组重复以及绑定嵌套重复范例处理变量绑定的编译。绑定嵌套范例还将正则表达式扩展成嵌套正则表达式。

尽管未在表I的简化树语法句法中示出，但根据本发明的非终止定义可具有正则参数和类型参数。在一个实施例中，编译器将正则参数作为继承属性处理，并使用绑定应用和绑定应用重复转换处理类型参数。

根据本发明变换成MFST可使用定义MFST的状态机图以示例性方式描述。在示意图中，转移被如下标记：b/α，其中 b 是所消费的输入符号，而 α 是所执行的动作。ϵ/α 表示不消费输入但执行某种动作的转移。这类转移被称

为动作转移。在执行变换机时，在本发明的一个实施例中，实现最大行进策略（maximal progress policy），它向自一状态的动作转移分派低于自该状态的所有输入消费转移的优先级。在示意图中，特殊符号'['以及']'分别表示输入集合的开始和结束。变量绑定被写成 $x =$ 表达式，并应用于变换机的当前环境。

对重复运算符，即'+'、'*'、和'?'，结果的集合使用列表累积，但可推广该机制以覆盖将结果累积成任何集合类型，例如通过使用相应的联接运算符。

根据本发明的各个实施例，按以下两种方式扩充绑定引用和绑定应用重复范例：内联和调用。内联扩充以空间交换时间，因此在本发明的一个非限定性的实现中，当确定化 MFST 时内联扩充是优选的。一般，调用扩充在三种情况中使用。第一种情况是当编译以便解释时。在这种情况中，编译器不确定化 MFST，并将所有的引用扩充为调用。第二种情况是递归。例如，如果 T Ref 引用了起始符号，则编译器将引用扩充为调用。第三种情况是扩展。如果 T Ref 引用了不透明的 MFST，则编译器将引用扩充成调用。

本发明的编译器为称为变换引擎（XE）的虚拟机将树语法转换成指令。XE 支持以下：（1）变换机的定义和操作，（2）控制转移到其它变换引擎实例以及（3）对环境的管理和访问。

关于第一类别，用于保存和解释状态机定义的技术是已知的。从而，以下描述的是用于实现第二和第三指令类别的示例性、非限定性方法：转移和管理。根据本发明，这些指令被定义为一对接口中的方法。编译器不直接生成对这些接口的调用；相反，编译器生成由 XE 实现解释的 XE 指令。

根据本发明提供的接口 XEControlInstructions（控制指令）在图 2A 的示例性伪代码 200 中示出。XEControlInstructions 接口 200 令当前 XE 实例作为隐式的操作数，即 XEControlInstructions 接口 200 的“该指针”。

在 QFX 的示例性实施例中提供的 Mark（标记）方法在标记栈上的输入项中记录当前位置。Yield（产生）方法从标记栈弹出标记 M，并返回输入项在 M 与当前位置之间的部分。在一个实施例中，QFX 经由遍历供应商接口间接实现这些方法。这样的方法有利地将项表示和遍历与项变换分开。遍历供应商因此可通过将 QFX 变换框架应用于任何特定的数据表示来实现遍历。

根据本发明提供的 Call（调用）方法创建新环境 E，并在 E 中保存

continuation（延续）和 callingEnvironment（调用环境）（见例如图 2A 的 XEControlInstructions 接口 200）。Call 方法然后返回将控制转移给 target（目标）的 continuation。Call 方法也调用 Mark 方法以保存当前输入位置。Return（返回）方法逆转该操作，并用于使用 Yield 方法将所匹配的项目保存在当前环境(E)中：E.term=Yield()。然后，Return 方法在变量 callingEnvironment. result（调用环境的结果）中存储 E；最后，Return 方法返回 continuation。当被调用时，continuation 将当前环境设置成 callingEnvironment，并从调用状态继续。

NewEnvironment（新建环境）方法根据本发明创建新环境。此外，图 2B 和 2C 的示例性、非限定性伪代码 210 和伪代码 220 分别概述了 PushEnvironment 和 PopEnvironment 方法的操作。

PushEnvironment 方法在内部环境栈上保存当前环境，并创建例如名为 tempEnv（临时环境）的新环境。PushEnvironment 然后将 tempEnv 绑定到 variableName（变量名），将当前环境设置成 tempEnv，并调用 Mark 来记录当前输入位置。

PopEnvironment 方法在变量“term（项）”中记录自前一 Mark 调用以来匹配的子项。PopEnvironment 然后从环境栈中还原当前环境。PopEnvironment 不保存弹出的环境，因为对 PushEnvironment 的相应调用已绑定所弹出的环境。

另外，Exec（执行）方法根据本发明执行动作。在 QFX 的示例性实现中，ActionRefernce（动作引用）是对应于方法指针的单独保存和加载的表的表索引。

根据本发明提供的 XEEnvironmentInstructions（环境指令）接口在图 2D 的示例性伪代码 230 中示出。XEEnvironmentInstructions 接口 230 令环境实例为显式的操作数，即 XEEnvironmentInstructions 接口 230 的“该指针”。

ChildEnv（子环境）方法创建其父亲为该指针的“该环境”的新嵌套环境。Bind（绑定）方法将变量绑定到值，如果绑定成功则返回真。在示例性、非限定性实施例中，如果 variableName 具有当前绑定，则 Lookup（查找）方法将值设置成该绑定并返回真。否则，它返回假。

本发明的变换引擎（XE）的操作支持向状态转移分派优先级。编译器通过对每一状态上可能的转移定序而解决因模式之间的选择引起的歧义。在一个

非限定性实施例中，对正常输入的所有转移优先于通配符转移，而后者优先于动作转移。而且，在这些优先级组中，程序员可任选地向特定转移分派优先级。

XE 实例可通过支持其中它们使用所有适用的转移以从一状态继续并按优先级次序对延续部分排队的模式来解决多个结果的生成。

图 3A 示出了根据本发明定义的转换范例的增强集的 αT 范例的示例性转换。更具体地，图 3A 示出了动作正则表达式范例的转换。该转换是嵌套正则表达式序列范例的转换的应用。正则表达式动作范例未示出，因为它是对序列范例的直接应用。在示意图中，圆圈 SC1、SC2 和 SC3 表示起始状态，圆圈 IC1、IC2 和 IC3 表示中间状态，而双圆圈 DC1、DC2 和 DC3 表示接受状态。

图 3B 示出了绑定嵌套范例 x : [T]的示例性转换。在图 3B 中，方法 PushEnvironment 和 PopEnvironment 被分别缩写成 PushEnv 和 PopEnv。起始状态 SC4、中间状态 IS3 和 IS4 以及接受状态 DC4 示出了图 3B 定义的机器的不同状态。在读取[之后，机器创建新环境 E，并将其绑定至 x。机器然后推入当前环境 C，并令 E 为当前环境。机器也标记当前输入位置。然后，机器执行 T，绑定 E 中的变量。在 T 执行期间，变量绑定在 E 中累积。机器然后在 E.term（环境项）中存储[T]匹配的项，并还原 C 作为当前环境。

图 3C 示出了绑定嵌套重复范例 x : [T]*|+|?的示例性转换。图 3C 示出了编译器如何转换带有重复的嵌套正则表达式。起始状态 SC5、中间状态 IS5、IS6、IS7 和 IS8 以及接受状态 DC5 示出了图 3C 定义的机器的不同状态。可见，由状态 IS5、IS6、IS7 和 IS8 状态定义的变换机的内在部分与图 3B 中的机器相同。在该机器周围的是容器状态机，包括起始状态 SC5 和接受状态 DC5。容器机器的目标在于累积环境作为绑定至变量 x 的列表。

就此方面，变换机创建空列表并将其绑定至 x。接着，变换机将执行 a[T]的结果追加到 x。执行和追加步骤通过标记为“对'*'和'?'使用该弧”的正向∈转移而可任选地呈现。执行和追加步骤通过遍历标记为“对'*'和'+'使用该弧”的反向∈转移而可任选地重复。

图 3D 示出了 x : (T) 范例的示例性转换。起始状态 SC6、中间状态 IS9 和 IS10 以及接受状态 DC6 示出了图 3D 定义的机器的不同状态。图 3D 示出了与图 3B 的绑定嵌套范例的变换机类似的变换机，除了图 3D 中进出 T 的转移不

消费输入符号。

图 3E 示出了 x : T Ref 范例的示例性转换，即使用调用扩充。图 3E 定义了包括起始状态 SC7、中间状态 IS11 和接受状态 DC7 的机器，并示出了对扩充 x : T Ref 的调用策略的使用。第一转移执行了调用指令 Call(T)，这是以下示例性、非限定性伪代码的简写：

Call(T, CurrentContinuation（当前延续）, CurrentEnvironment（当前环境）)

调用指针处的 CurrentContinuation 是转移箭头尾部的中间状态 IS11。所调用的变换机在变量结果中返回其环境。最后，进行调用的变换机将结果绑定至 x。

图 3F 示出了 x : T Ref*|+|?范例的示例性转换，即使用调用扩充。图 3F 定义了包括起始状态 SC8、中间状态 IS12 和 IS13 以及接受状态 DC8 的机器，并示出了应用于调用的重复容器。该变换机类似于图 3C 的变换机，采用对 T 的调用作为内在变换机以代替 T 的内联扩充。

在图 3A 到 3F 中没有描述的转换范例是绑定组重复范例和绑定引用范例的内联版本。绑定组重复组合了图 3F 的重复容器与图 3D 中所示的绑定组变换器。另外，绑定引用和绑定重复范例的内联版本分别与绑定组和绑定组重复范例相同。

根据本发明，确定化描述如何制定子集构造技术以虑及可能具有动作的转移。根据本发明，允许动作与转移和状态两者的关联。尽管可仅采用对转移的动作来构造确定性变换机，但该选择令算法更为复杂，而高效地为状态和转移两者存储动作是直接的。

简化某些细节，图 2E 列出了如在 QFX 中实现的示例性、非限定性∈闭包算法，即 State⟨T⟩.EClosure+。扩充该方法以累积对∈转移的动作。图 2E 的示例性伪代码 240 中标记为(A1)和(A2)的行在列表“actions（动作）”中保存在∈闭包构造期间遍历的对 epsilon（ε）转移的所有动作。总的确定化算法将这些所保存的动作转移到自∈闭包构造的确定性有限状态变换机（DFST）状态。

图 2F 示出了如按照 QFX 中的示例性方式所实现的被修改以容纳动作的子集构造，即 NFA⟨T⟩.SubsetConstruction。修改并添加图 2F 的示例性伪代码 250

标签(B1)到(B6)的标记行以容纳动作。行(B1)和(B2)找到源状态（src）具有对符号的转移的所有状态；这些行也在 symActions（符号动作）中累积每一转移的动作。行(B3)和(B4)执行∈转移的类似功能。最后，行(B5)将 symActions 添加到 src 与 dst（目的）状态之间的 DFST 转移，且行(B6)将 epsActions（ε 动作）添加到 src 状态。这些动作然后在完成到 src 的转移之后执行。在一个实施例中，这些算法假定动作除了在当前环境中绑定变量之外不会有侧放作用。

总体上，有利地，根据本发明的用于确定化的该方法实现以下保证：

1. 如果一对动作 α1 和 α2 是同一产生的元素，则动作 α1 和 α2 按词汇次序调用；

2. 如果动作 α1 是产生 p1 的一部分，而动作 α2 是产生 p2 的一部分，p1 ≠ p2，则变换机将调用 α1 或 α2 中的任一个。变换机仅在 p1 和 p2 共享前缀 *pre* 且 *pre* 包含 p1 中的 α1 且 *pre* 包含 p2 中的 α2 的情况下调用这两个动作；以及

3. 在语法的起始定义内，每一产生的最后动作具有侧放作用。在一个实施例中，变换机不会运行该动作，直到所有其它动作均完成。

在这一方面，属性化树语法允许程序员利用以上保证。使用属性语法，程序员可将所有的动作表达为当前环境内的变量绑定。这样的动作可采取两种形式之一：*x* : *T* 或 *x* = *表达式*，其中*表达式*可从当前环境读取值或调用代码来计算值。只要表达式执行的代码不引入次序依赖关系，执行树语法的结果就是确定性的。

程序员可使用属性语法来推迟侧放作用，直到明确地该侧放作用应被调用。例如，为了遍历树打印文本，程序员可使用属性将文本收集成串，然后根据与树语法起始符号相关联的最后动作打印该串。程序员也可通过使打印动作遵循明确的非终止定义而选择“在运行时”打印子树。

对于根据本发明的示例性、非限定性实施例的变量重命名，为确保变量绑定动作的独立性，编译器重命名变量，使得 MFST 中绑定的每一变量是唯一的。如果编译器也为给定动作生成代码，则编译器也重命名 α 中的变量。如果代码块 B 对于编译器是不透明的，则编译器安排向 B 传递间接通过原始变量名更新后的名字来查找其原始名字的环境。

可任选地，可通过执行活动范围分析和令若干原始变量共享同一更新后的名字来优化变量重命名。而且，变换机性能可通过使用编译器确定的偏移量而将变量引用实现为数组访问来改进。

从而，在一个方面中，本发明提供用于执行高效树变换的一般框架，该变换跨一组不同的 FST 表示实现对在其表示中支持动作信息的 FST 的动作语义的保存。可对 FST 执行任何交、并和补变换运算以及带有组合的其它布尔运算符，同时保存动作语义。

此外，如以上在背景中所提及地，在有限的环境下，存在可跨树变换保存有序嵌套信息的某些系统以及可跨树变换保存无序嵌套信息的某些系统，但尚不存在可跨例如交、并、补等树变换保存有序和无序树信息两者的系统。从而，在各个非限定性实施例中，用于执行高效树变换的框架还在执行 FST 的树变换的同时保存有序和无序嵌套信息。

有序和无序信息之间的区别在图 4A 和 4B 中概念性示出。图 4A 示出了也被称为列表模式的有序模式，它要求红、然后绿、然后蓝、然后再红的序列以便匹配模式。从而，在模式匹配意义上，考虑了模式元素出现的次序。图 4B 示出了也被称为集合模式的无序模式。图 4B 的集合模式示出了与图 4A 的列表模式相同的元素，但这次没有任何次序。图 4B 的集合模式因此表示了以任何次序的两个红、一个绿和一个蓝的集合。当进行模式匹配时，考虑的是元素在集合中的出现，即元素是否出现在两个树中，而非元素以何种次序出现。

计算系统中列表模式匹配情形的实例可以有口令的输入，其中口令是数字的顺序集合。由于为口令输入的每一字符必须以特定次序输入以便匹配为该用户存储在系统中的正确口令，口令匹配情形将基于具有有序信息的树结构匹配模式。

计算系统中集合模式匹配情形的示例有在文件系统中搜索一组指定的文件，例如 Pic_Amy、Pic_Greg、Pic_Neyda。当搜索这些图片出现的文件夹时，匹配的是它们在文件夹中的出现而非它们以何种次序出现。换言之，搜索情形中的用户仅关注找到具有这些文件中的每一个的文件夹，而图片在系统中存储的次序对用户而言是不重要的。集合模式匹配情形的另一示例是在数据库中一起找到某一名、中间名和姓，结果不依赖于数据是被存储为“姓、名、中间名”、

“名、中间名、姓”还是“名、姓、中间名”。这三者全部以任何次序的任何出现都满足该数据库查询。

根据本发明的框架，可对 MFST 执行变换同时保存由树数据结构表示的有序和无序信息两者，包括任何交、并和补变换。因此可对组合关于树的孩子的集合和列表假设的模式执行变换。因此，嵌套在树结构的节点中的有序或无序信息均跨变换保存。

如图 5A 的流程图所示，在 500 本发明在计算系统中启用用于接收包括有序和/或无序信息的各种类型的有限状态变换机（FST）数据结构的一般框架，然后在 505 将框架的变换引擎应用于 FST。在 510，根据上述任何类型的 FST 表示的树语法进行 FST 的分析和变换。结果，在 515，FST 被变换以生成 MFST，这保存 FST 的动作信息，例如在对两个或多个 FST 执行任何交、补和并运算的同时保存有序和无序嵌套信息，并允许将非确定性 FST 变换成确定性结果。

图 5B 是大体对应于示出本发明的树语法所允许的这一方面的以上过程的框图。变换框架 530 接收分别包括有序信息、无序信息或两者的一个或多个 MFST 520、522 或 524，根据任何交、并和补变换运算（和/或其他布尔运算符，带有组合）变换 MFST，并将有序和/或无序信息保存在输出 540 中。

图 5C 到 5I 示出了在存在有序或标记信息的情况下对状态机的变换的示例性、非限定性的各方面。如图 5C 的框图中所示，问题涉及两个状态机 M_1 和 M_2 的并，状态机 M_1 和 M_2 各自具有单独的接受状态 550 和 555，即如何在机器的变换表示中表示该信息。 在以往，这是通过将两个接受状态 550 和 555 组合成表示两者的单个接受状态而实现的，这可导致来自原始机器的信息的损失。

根据图 5D 的框图，在示例性、非限定性方面中，本发明在结果变换机 560 中接受自家状态机标记的接受状态。在各个非限定性实施例中，本发明在变换期间引入对标记并 U_L 的处理，这是用接受状态所源于的机器的名字标记并机器的接受状态的并的变体。

如图 5E 的示例所示，当执行 $M_1\ U_L\ M_2$ 时，存在接受状态标签的三种可能：“M_1”575、“M_1, M_2”580 和“M_2”585。这指示标记并机器中的接受状态分别表示由 M_1、M_1 和 M_2 两者、以及 M_2 接受的输入。如在图 5I 中更详细地示

出，在示例性、非限定性实现中，这些标签可被示为位，每一原始机器一个位。例如，可向 M_1 分派低位，而向 M_2 分派高位。于是，可能的标签可被表示为“01”（即，M_1 而非 M_2 接受输入），“11”（即，M_1 和 M_2 均接受输入）以及“10”（即，M_2 而非 M_1 接受输入）。另一种可能性是“00”。并机器的均非接受状态可被视为具有该标签（因为 M_1 或 M_2 均不接受输入）。

在此方面，如图 5E 的框图中所示，除了生成表示 M_1M_2 组合的结果接受状态节点 580 以外，本发明也生成表示接受状态 M_1 和接受状态 M_2 的接受状态节点 575 和 585。如图 5F 中所示，当遇到并运算符 $M_1 \cup M_2$ 时，这些接受状态 575、580 和/或 585 中的任一个可以是得到的接受状态的一部分，而如图 5G 中所示，交运算符 $M_1 \cap M_2$ 仅意味着 M_1M_2 节点 580。例如，当向 M_1 分派低位而向 M_2 分派高位时，在一个非限定性实施例中，并机器被通过找到标记为“11”的接受状态而被转换成交机器。如果不存在这样的状态，则交机器为空。给定标记为“11”的一组状态 A，通过从并机器移除不能从其到达 A 中任何状态的任何状态而生成交机器。

如图 5H 的框图所示，也可在根据本发明的变换期间捕捉子类型化。关于类型化存在可在确定化组件 570 时确定的三种结果。$M_1 \subset M_2$ 是如果没有接受状态被标记为 M_1 的一个结果，如顶部所示。$M_1 = M_2$ 是如果在中间仅生成接受状态 M_1M_2 580 时的另一结果。$M_2 \subset M_1$ 是如果没有接受状态被标记为 M_2 的一个结果，如在底部所示。给定以上变换过程，可确定子类型化。例如，当向 M_1 分派低位而向 M_2 分派高位时，如果所有接受状态都被标记为 11，则 M_1 等于 M_2。如果存在标记为 10 和 11 的接受状态，则 M_2 包含 M_1（即，M_1 是 M_2 的子类型）。

图 5I 示出了根据标记并运算符变换机器 590 和机器 592 的示例性、非限定性过程的流程图，其中两个机器 590 和 592 中的每一个是两位机器。根据应用位标签的上述示例性、非限定性实现，向机器 590 的标签 L_1 和 L_2 分派两位，并向机器 592 的标签 L_3 和 L_4 分派两位。当根据本发明的标记并运算符组合时，这在结果 4 位机器 595 中产生 16 个接受状态。例如，L_2 将接收标签 0010，L_2L_4 接收标签 1010，$L_1L_2L_3L_4$ 接收标签 1111，依此类推，直到表示了所有组合。

图 5J 示出了如何变换有序机器以构造和组合相应的无序机器。例如，模

式集合 P 可能包括可任选出现的约束，诸如由以下集合{p_1, p_2*, p_3^+}表示的约束，这指示包括至少一个 p_1 的集合、包括零或多个 p_2 以及一个或多个 p_3 的集合。根据本发明的示例性、非限定性实现，集合模式 P 被分成两个组成部分：（A）并模式 596，诸如在目前描述的示例集合 P 中的 $p_1 \cup p_2 \cup p_3$，以及（B）标记运行的集合 598，其各自接受某些输入。在目前所述示例中，这可得到{L_1, L_2*, L_3^+}，其中 L_1、L_2 和 L_3 分别标记接受状态 p_1、p_2 和 p_3。

从而，如图 5K 所示，为根据本发明的示例性、非限定性实现对诸如给定无序模式 P1 和 P2 等无序模式执行标记并 U_L，计算以下表达式：$\cup(P_1)$、$\cup(P_2)$、$\Gamma(P_1)$和$\Gamma(P_2)$，其中$\cup(x)$是 x 的并模式，而$\Gamma(x)$是 x 的接受运行的集合。于是，基于这些表达式，获得以下两个结果：

(1) $\cup(P_1 \cup_L P_2) = \cup(P_1) \cup_L \cup(P_2)$

(2) $\Gamma(P_1 \cup_L P_2) = \Gamma(P_1) * \Gamma(P_2)$

其中图 5K 以及以上等式(2)中的的“*”运算符 LUCP 表示标记并叉积运算。

接着，重新标记$\cup(P_1 \cup_L P_2)$，使得运行标记是一致的。例如，通过重写 R 的冗余标签组中的每一个 *l* 来实现重新标记。从 R 中，选择任意成员 z（例如，上述按标签的位的实现中的编号最低标签或次序最低位），且对 R 中的每一 *l* 将 *l* 重写成 z。例如，如图 5L 中所示，其中?(P_1 ?$_L$ P_2)具有示例性接受状态标签 ASL1，包括“L_1,L_2”、“L_1,L_3,L_4,L_2”和“L_2,L_4”，则 L_3 可由 L_1 替换，而 L_4 可由 L_2 替换以形成压缩接受状态标签 ASL2，即“L_1”、“L_1,L_2”和“L_2”。

如上所述，图 5K 以及以上等式(2)中的“*”运算符 LUCP 表示标记并叉积运算。在如上图所述重新标记了 P_1 和 P_2 的接受运行之后，可根据以下非限定性过程定义标记并叉积运算。首先，为一对运行 R_1 和 R_2 定义运算符 U_L。为此，标签(R)被定义为运行 R 的标签。R 然后被定义为对(*l*, [x, y])的集合，其中 *l* 是来自重新标记的 $P_1 \cup_L P_2$ 的某一标签，而 x 是 *l* 的最小出现数，而 y 是 *l* 的最大出现数，且$[x, y] \subseteq [0, \infty]$。通过考虑 $R_1 * R_2$（即，R_1 和 R_2 的叉积）中的每一元素对来计算 $R_1 \cup_L R_2$，如下。对每一对(l_1, [x_1, y_1])、(l_2, [x_2, y_2])，如果 $l_1 = l_2$，则(l_1, [x_1, y_1]), (l_2, [x_2, y_2])使用区间并运算变为(l_1, [x_1, y_1]) U (l_2, [x_2, y_2])。否则，(l_1, [x_1, y_1]), (l_2, [x_2, y_2])是空集。

因此，对 $R_1 * R_2$（即 R_1 和 R_2 的叉积)中的成员 z，U_L 被计算为或者是(l_1, $[x_1, y_1]$) U (l_2, $[x_2, y_2]$) 或者是空集。然后通过取得这些结果的并来形成结果集 S 而计算 $R_1 U_L R_2$。如果 S 具有与 R_1 和 R_2 相同的基数，则 $R_1 U_L R_2$ 是带有例如标签(R_2)和标签(R_2)等标签的 S。否则，$R_1 U_L R_2$ 是{R_1, R_2}，其中 R_1 和 R_2 不相交，且它们具有单独的标签。

例如，如果：

$R_1 = \{(L_1, [0, 1]), L_2, [1, \infty]), L_3, [1, 1])\}$

$R_2 = \{(L_1, [1, 2]), L_2, [2, 2]), L_3, [1, 1])\}$

则 $R_1 U_L R_2$ = 带有标签标签(R_1)、标签(R_2)的$\{(L_1, [0, 2]), L_2, [1, \infty]), L_3, [1, 1])\}$。

又例如，如果：

$R_1 = \{(L_1, [1, 1]), L_2, [2, \infty]), L_3, [1, 1])\}$

$R_2 = \{(L_1, [0, \infty]), L_2, [1, 1]), L_3, [4, 4])\}$

则 $R_1 U_L R_2 = \{R_1, R_2\}$ 因为 R_1 和 R_2 不相交。

又例如，如果：

$R_1 = \{L_1, L_2\}$ 其中 L_1 是(L_1, [1, 1])等

$R_2 = \{L_1, L_2, L_3^*\}$

则 $R_1 U_L R_2$ = 带有标签标签(R_1)、标签(R_2)的$\{L_1, L_2\}$以及带有标签标签(R_2)的$\{L_1, L_2, L_3^+\}$。

总而言之，根据上述过程首先计算 $R_1 U_L R_2$，即根据本发明的运行 R_1 和 R_2 的 U_L 的定义。然后，通过取得 R_1 与($R_1 U_L R_2$)的交以及 R_2 与($R_1 U_L R_2$)的交即 $R_1 \cap (R_1 U_L R_2)$和 $R_2 \cap (R_1 U_L R_2)$来细化结果，即 $R_1 \cap (R_1 U_L R_2)$和 $R_2 \cap (R_1 U_L R_2)$是使用类似于并运算符 U_L 的方法执行的，只是用区间交代替了区间并运算。

在本发明的示例性、非限定性实施例中，为此，并运行被分成片段 R_1、R_2 和 $R_1 \cap R_2$。这些片段 R_1、R_2 和 $R_1 \cap R_2$ 然后分别用标签(R_1)、标签(R_2)和标签(R_1)，标签(R_2)标记。

从而，其中 P_1 和 P_2 是无序模式，可执行 $P_1 U_L P_2$ 的确定化。就此而言，本发明使得 $P_1 U_L P_2$ 能被计算成带有并模式部分以及并模式的接受运行的

集合的无序模式，每一运行标记了来自 P_1、P_2 或两者的标签。$P_1 U_L P_2$ 于是可以是与某个其它无序模式 P_3 的并，该模式也具有并模式以及标记运行的集合，对另一无序模式 P_4 依此类推。

示例性、非限定性使用情形

为了补充理解，本发明的各个使用情形示出了可在计算系统中应用使用本发明的 Q 框架的模式匹配的各种应用。匹配同时保存嵌入正在变换的树中的动作语义或有序和无序信息的能力由此在计算系统中启用了各种模式匹配系统的集合。然而，所选的实际情形仅是示例性的，从而不作为对本发明适用的模式匹配的领域的限制。实际上，根据本发明变换 MFST 的能力，按定义是相当广泛的，因为无需顾及丢失关于动作约束的信息或嵌入正在变换的树表示中的有序和无序信息。

就此方面，如上所述，Q 框架允许在根据至少交、并和补变换而变换 MFST 时保存这样的动作或有序和无序的信息。采用这三种变换运算，可执行树数据结构上的一性质，其被本领域的技术人员称为结构兼容性或子类型化。通常，计算应用程序想要知道给定树是否是另一给定树的子集或树的集合（或并、补或交所定义的这些树的某种变换运算）。

本发明可应用的另一类重要情形是执行静态类型检查的大量应用程序，其中编译器查看程序，并要求过程兼容性。过程兼容性检查涉及树集合的模式匹配，且有助于找出计算机程序中的隐错。

一般而言，用户或计算系统可能希望针对树数据集合或树数据的子集实现的测试类是无限的。然而，通常会发生某些递归测试。例如，通常测试希望知道第一树或第一树集合是否与第二树或第二树集合相交。或者，可能希望知道是否存在树集合的合计，以及是否存在对树遍历的所有路径的覆盖。或者用户可能希望知道什么树导致默认情况。还可希望测试空测试，这询问了是否可接受任何树作为模式匹配的问题。如前所述，在本发明的一个实施例中，本发明允许以下四种变换测试：交、并、补和空测试。

以上描述了采用编译器对本发明的有利使用。本发明也可用于各种其它情形，诸如模式确认。例如，消息随购买定单而来，问题是购买定单是否兼容某

种模式。本发明的模式匹配可用于针对模式确认消息，而不考虑动作语义或有序/或无序信息而同时在结果中保存动作语义或有序/无序信息。

契约检查是本发明的另一示例性用途。例如，公司可能具有涉及公司的人力资源计算系统的策略，它规定它必须遵守一组表示物理系统的计算机要求（例如，X 数量的存储、Y 数量的安全、Z 数量的处理能力等）。就此方面，每一配置可被表示为树表示中的一组要求，且根据本发明的框架的树变换能力，系统分析者可根据预定义变换来变换计算机配置，并查看新的计算机配置是否匹配 HR 计算系统所需的契约。

本发明还可应用于找到计算系统所记录的日志数据集合中的安全漏洞模式，或确定计算系统的可能配置。本发明可应用于防火墙，在那里传入和传出防火墙的消息可被视为树，其中可采用模式匹配来观察是否应不让任何给定消息通过防火墙。由于本文提及的、Q 框架所支持的变换布尔运算可用于形成任何逻辑语句，计算系统中的任何规则系统可被归约成树，且可应用模式匹配来确定是否遵守这些规则。因此，应清楚本发明的应用是不受限制的。

有限状态自动机和变换机

对附加的上下文，有限状态机（FSM）或有限状态自动机（FSA）是由状态、转移和动作组成的行为的模型。FSM 的状态存储关于过去的信息，即状态反映从系统起始到目前时刻的输入改变。转移指示状态改变，且由要求履行以允许转移的条件描述。动作是要在给定时刻执行的活动的描述。存在若干动作类型：进入、退出、输入和转移动作。进入动作在进入状态时执行该动作。退出动作在退出状态时执行该动作。输入动作依赖于目前状态和输入条件执行动作。转移动作在执行某一转移时执行该动作。

有限状态变换机（FST）是使用动作基于给定输入和/或状态生成输出的一类 FSM，且可用于控制应用程序、构造计算机程序等。图 6A 示出了简单的有限状态变换机 620。变换机 620 基于给定输入 610 生成输出 630，如由 FSM 620 所转换或变换地。存在的两类变换机 FSM 是 Moore 模型和 Mealy 模型，它们将在以下更详细描述。也常使用混合模型。

因此，可理解可将众多不同种类的计算机系统和进程建模成 FSM 和 FST。例如，可使用有向图将任何可扩展标记语言（XML）文档表示成 FSM。关系

数据库中的关系数据也可按照这种方式表示，例如借此输入（例如，查询）由FST转换成输出（例如，查询结果），FST表示底层关系存储。一般而言，当用于对计算机进程建模时，FST通常被表示为标记边有向图，其中每一顶点表示n个状态之一，而每一边表示在接收标记该边的字母符号时从一个状态到另一个状态的转移。

当计算机系统中的众多复杂子系统和进程被通信耦合成总系统的一部分时，为了设计总系统的软件，设计者可首先将每一子系统和进程表示为FST，例如有向图或其它等效表示。然后，为了为连接不同子系统的单个系统创建复杂计算机程序，例如有向图的FST可根据各种操作组合或以其它方式变换以便形成表示总系统的行为的新的有向图。

例如，假定用户 Jane 经由因特网从客户计算机向因特网服务器作出对存储在数据库中的朋友 John 的度假照片的请求，并得到因特网服务器上的应用程序的服务。如可以理解地，根据这样的请求进行的端对端通信是数目众多的，以对Jane的认证开始并确保除了执行请求本身以外Jane还被授权查看John的照片。作为简单的示例，作出和处理请求本身可被建模成第一FST。在服务器处，第二FST可对参考各种规则检查Jane是否是John的朋友的行为建模，这些规则诸如可在访问控制列表（ACL）和相应的策略的集合中找到，它们可被表示成一连串XML片段、树或有向图。另外，第三FST可对关系数据库本身建模。通过对第一、第二和第三FST集合进行变换、组合、匹配、转换等，可形成新的有向图，它表示系统并对特定的输入请求返回是（“授权”）或否（“未被授权”）的回答，且处理照片的传递。

一般而言，FSM可使用如图8A的简单状态转移图的状态图（或状态转移图）表示。在图8A中，表示了分别具有进入动作进入A1和进入A2的两个状态S1和S2，意味着在进入状态S1时执行进入动作进入A1，而在进入状态S2时执行进入动作进入A2。而且，当转移条件TC1发生时，发生自状态S1到S2的转移T1；而当转移条件TC2发生时，发生自状态S2到S1的转移T2。状态S1可能是“门打开”，而状态S2可能是“门关上”。从而，为了从状态S1转到状态S2，必须发生转移T1，这仅当转移条件TC1发生时才发生，而TC1可能是“力正使得门在关闭方向上移动”。在进入状态S2时执行的进入

动作进入 A2 因此可能是“关上门”。在打开门时可跟随类似的转移链，即从状态 S2 转到状态 S1。

除了示意图之外，还可使用不同类型的状态转移表来表示 FSA。这样的状态转移表 STT1 的常见表示在图 7A 中示出，其中诸如当前状态 B 等表 STT1 的列与诸如状态 Y 等表 STT1 的行的组合指示处于状态 B 时当发生条件 Y 时发生的下一状态，即状态 C。然而，采用诸如状态转移表 STT1 的表，仅可使用脚注来添加完整的动作信息。

然而，存在使用状态表包括全部动作信息的 FSM 定义。例如，在虚拟环境中定义的 FSM 被称为虚拟有限状态机（VFSM），它涉及使用输入控制性质和输出动作的分派名字描述控制系统的行为的软件指定方法。

虚拟环境表征其中 VFSM 操作的环境，且由三个名字集合定义：输入名、输出名和状态名。输入名由所有可用变量的控制性质表示。输出名由对变量的所有可用动作表示，而状态名为 FSM 的每一状态定义。输入名用于构建执行状态转移或输入动作的虚拟条件。虚拟条件是使用正逻辑代数构建的。输出名用于触发动作（进入动作、退出动作、输入动作或转移动作）。

状态表定义 VFSM 状态行为的细节，如图 7B 的示例性状态表 ST1 所示。状态转移表 ST1 包括三列：在第一列中，使用状态名 SN；在第二列中，放置了使用正逻辑代数根据输入名构建的虚拟条件 CO；在第三列中，出现用于触发动作 AC 的输出名。

除了如在此处呈现的其对反应系统建模的用途，FSA 在众多不同领域中是重要的，包括语言、计算机科学、哲学、生物学、数学和逻辑。有限状态机是在自动机理论和计算理论中研究的一种类型的自动机。在计算机科学中，有限状态机广泛用于对应用程序行为建模、硬件数字系统的设计、软件工程、编译器以及计算和语言的研究。对 FSA 应用的完整审视实际上是不可能的——存在将 FSA 应用到系统中任何地方的实际上无限的应用。

一般而言，变换机计算两种形式语言之间的关系。在 FSM 和 FSA 的上下文中，变换机使用动作基于给定输入和/或状态生成输出，且可用于控制应用程序。一般区分两种类型的变换机 FSM：Moore 模型和 Mealy 模型。实际上，常使用混合模型。

为说明性的目的，图 8B 示出了变换机 FSM 800，它示出具有四个状态 S3、S4、S5 和 S6 以及相应的输出 O3、O4、O5 和 O6 的 Moore 模型示例。采用 Moore 机器，FSM 仅使用进入动作，即输出仅依赖于状态。Moore 模型的优点在于行为的简化。图 8B 中的示例示出了电梯门 Moore FSM 800。状态机识别两个命令："命令开" C1 和"命令关" C2，它们触发状态改变。例如，在"正在打开"状态 S6 中的进入动作 EA1 启动打开门的电机，而"正在关闭"状态 S4 中的进入动作 EA2 启动关闭门的另一方向上的电机。"已打开"状态 S3 和"已关闭"状态 S5 在此示例中不执行任何动作，相反它们分别向外部系统（例如，其它状态机）用信号表示"门打开"或"门关上"。

图 8C 示出了变换机 FSM 810，它示出了具有两个状态 S7 和 S8、相应的输出 O7 和 O8 以及输入动作 I1 和 I2 的 Mealy 模型示例。采用 Mealy 机器，FSM 仅使用输入动作，即输出依赖于输入和状态。对 Mealy FSM 的使用通常导致状态数目的减少。图 8C 中的示例示出实现与图 8B 的 Moore 示例 800 相同行为的 Mealy FSM 810.存在两个输入动作："如果命令关到达，则启动电动机以关闭门" I1 和"如果命令开到达，则启动另一方向上的电动机以打开门" I2。

有限自动机的另一区分是确定性有限自动机（DFA）和非确定性（NDFA）或通用非确定性有限自动机（GNFA）。在确定性自动机中，对于每一状态，对每一可能的输入存在恰好一个的转移。在非确定性自动机中，对给定可能的输入，自给定的状态可能存在零个或一个以上的转移。这种区分是关乎实际，而非理论的，因为存在可将任何 NDFA 变换成等效的 DFA 的算法，尽管这种变换通常显著增加自动机的复杂性。

仅具有一个状态的 FSM 被称为组合 FSM，且仅使用输入动作。这一概念在其中需要多个 FSM 来一起工作，且便于将纯粹组合的部分考虑成一种形式的 FSM 以适合设计工具的情况中有用。

一般而言，变换机计算两种形式语言之间的关系。有限状态变换机（FST）计算的一类关系被称为有理关系类。FST 通常有助于自然语言处理研究。

FST 是具有两个带的有限状态机，它与具有单个带的普通有限状态自动机形成对比。关于命名，将自动机说成识别串，如果其带的内容被视为输入。换言之，自动机计算将串映射到集合{0, 1}的函数。或者，将自动机说成生成串，

这意味着其带被视作输出带。根据这一观点，自动机生成作为串的集合的形式语言。自动机的这两种观点是等价的：自动机计算的函数正是它所识别的串集合的指示函数。有限自动机生成的语言类被称为正则语言类。

变换机的两个带一般被视为输入带和输出带。就此而言，将变换机说成通过接受其输入带上的串并在其输出带上生成另一串而将其输入带的内容变换（即，转换）成其输出带。它可非确定性地这样做，且它可能会对每一输入串产生多于一个的输出。变换机也可能对给定输入串不产生输出，在这种情况下，将它称作拒绝输入。

对于附加的上下文，在形式上，有限状态变换机 T 是多元组(Q, Σ, Γ, I, F, δ)，使得：

Q 是有限集，状态的集合；

Σ 是有限集，被称为输入字母表；

Γ 是有限集，被称为输出字母表；

I 是 Q 的子集，初始状态的集合；

F 是 Q 的子集，最终状态的集合；以及

$\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \times Q$ （其中ϵ是空）是转移关系。

(Q, δ) 可被视为标记有向图，被称作 T 的转移图：顶点的集合为 Q，且 $(q, a, b, r) \in \delta$ 意味着存在自顶点 q 到顶点 r 的标记边。在此方面，a 是该边输入标签，而 b 是输出标签。

将扩展转移关系 δ *定义成使以下公式成立的最小集合：

$\delta \subseteq \delta^*$；

对所有的 $q \in Q$，$(q, \epsilon, \epsilon, q) \in \delta$；以及

只要 $(q, x, y, r) \in \delta^*$ 且 $(r, a, b, s) \in \delta$，则 $(q, xa, yb, s) \in \delta^*$。

扩展转移关系实质上是已经扩充来考虑边标签的转移图的自反传递闭包。δ *的元素被称为路径。路径的边标签通过按次序串接其组成转移的边标签来获得。

变换机 T 的*行为*是如下定义的有理关系[T]：x[T]y 当且仅当存在 $i \in I$ 和 $f \in F$ 使得 $(i, x, y, f) \in \delta^*$。即如果存在自输出状态到最终状态的输入标签为 x 而输出标签为 y 的路径，则 T 将串 $x \in \Sigma^*$ 变换成串 $y \in \Gamma^*$。

对有限自动机定义的以下运算也适用于有限变换机：并、串接、克林闭包、组合、输入带的投影以及输出带的投影。

对于并运算，给定变换机 T 和 S，存在变换机 $T \cup S$，使得 $x[T \cup S]y$ 当且仅当 $x[T]y$ 或 $x[S]y$。

对于串接运算，给定变换机 T 和 S，存在变换机 $T \cdot S$，使得 $wx[T \cdot S]yz$ 当且仅当 $w[T]y$ 或 $x[S]z$。

对于克林闭包运算，给定变换机 T，存在具有以下性质的变换机 T^*：（1）$\varepsilon[T^*]\varepsilon$；（2）如果 $w[T^*]y$ 且 $x[T]z$，则 $wx[T^*]yz$；且 $x[T^*]y$ 不成立除非由（1）或（2）委托。

注意到不存在变换机的交的概念。相反，存在组合运算，这是变换机专用的，且其构成类似于自动机的交。如下定义组合：

给定字母表 Σ 和 Γ 上的变换机 T，以及字母表 Γ 和 Δ 上的变换机 S，存在 Σ 和 Δ 上的变换机 $T \circ S$，使得 $x[T \circ S]z$ 当且仅当存在使得 $x[T]y$ 且 $y[S]z$ 的串 $y \in \Gamma^*$。

也可将变换机的任一带投影来获得自动机。存在两种投影函数：π_1 保存输入带，而 π_2 保存输出带。第一投影即 π_1 被定义如下：

给定变换机 T，存在有限自动机 $\pi_1 T$，使得 $\pi_1 T$ 接受 x 当且仅当存在 $x[T]y$ 的串 y。类似定义第二投影即 $\pi 2$。

此外，有限状态机可用于表示偏序，这形成了集合元素的定次序、定顺序或排列的直观概念。偏序不必是全序，这保证了集合中所有元素可互相比较。就此方面，全序是为集合项目的所有对定义的一种偏序。

因此，对某些但不必是全部项目对定义偏序。例如，集合{a, b}和{a, c, d}是{a, b, c, d}的子集，但它们不是彼此的子集。因此“子集”是集合上的偏序。作为另一示例，≤（小于或等于）是整数上的全序，因为对任何两个整数，整数之一总是小于或等于另一个整数。

如图 6B 中所示，有序嵌套表示“a”“隐含”“b”的信息，但根据该有序分层结构周围不存在其它方式（即，“b”不必“隐含”“a”）。因此，图 6B 的树是有序嵌套的表示的示例。相反，图 6C 的树表示无序嵌套的示例。在此方面，无论图 6C 的树表示的事实信息是从左向右读取的还是从右向左读取

的，都收集了同样的事实信息，即“块等于（是）红”或“红等于（是）块”在逻辑上均指示块是(=)红。从而，这样的信息是无序的，且遍历树的次序对于获取无序信息而言不是重要的。

示例性网络化和分布式环境

本领域的普通技术人员可以理解，本发明可以结合任何计算机或可作为计算机网络的一部分来部署的其它客户机或服务器设备来实现，或可在连接至任何种类的数据存储的分布式计算环境中实现。在这一点上，本发明涉及任何计算机系统或环境，其具有任意数目的存储器或存储单元，以及发生在任意数目的存储单元或卷上的任意数目的应用程序和进程，它们可结合根据本发明的 QFX 的实施例来使用。本发明可应用于具有部署在具有远程或本地存储的网络环境或分布式计算环境中的服务器计算机和客户计算机的环境。本发明也可应用于具有编程语言功能、用于生成、接受和发送关于远程或本地服务和进程的信息的解释和执行能力的独立计算设备。如前所述，MFST 普遍适用于多个机器和计算设备上的软件进程，因此根据本发明用于将语法转换成 MFST 的技术可高效地在各种计算环境中应用。

分布式计算通过计算设备和系统之间的交换提供了计算机资源和服务的共享。这些资源和服务包括信息的交换、对于诸如文件等对象的高速缓存存储和盘存储。分布式计算利用网络连接，允许客户机利用它们的集体力量来使整个企业受益。就此，各种设备可以含有其中蕴含本发明的 QFX 的应用程序、对象或资源。

图 9 提供了示例性的网络化或分布式计算环境的示意图。分布式计算环境包括计算对象 910a、910b 等，以及计算对象或设备 920a、920b、920c、920d、920e 等。这些对象可包括程序、方法、数据存储、可编程逻辑等等。这些对象可包括诸如 PDA、音频/视频设备、MP3 播放器、个人计算机等的相同或不同设备的各部分。每一对象可通过通信网络 940 与另一对象通信。该网络本身可以包括向图 9 的系统提供服务的其它计算对象和计算设备，且其本身可以表示多个互连的网络。根据本发明的一方面，每一对象 910a、910b 等，或 920a、920b、920c、920d、920e 等，可包含可利用适用于根据本发明的 QFX 的各个实施例的 API、或其它对象、软件、固件和/或硬件的应用程序。

还可以理解，诸如 920c 等对象可以主存在另一计算设备 910a、910b 等或 920a、920b、920c、920d、920e 等上。因此，尽管所示的物理环境可以将所连接的设备示为计算机，但是这样的图示仅是示例性的，并且该物理环境可以被替换地描述或描绘成含有诸如 PDA、电视机、MP3 播放器等的各种数字设备，它们中的任何一个可采用诸如接口、COM 对象等各种有线和无线服务、软件对象。

存在支持分布式计算环境的各种系统、组件和网络配置。例如，计算系统可以由有线或无线系统、本地网络或广泛分布的网络连接在一起。当前，许多网络耦合至因特网，后者为广泛分布的计算提供了基础结构并包含许多不同的网络。任何基础架构可用于便于本发明的 QFX 的实施例的示例性通信。

在家庭网络环境中，有至少四个全异的网络传输媒体，其每一个可支持一种唯一的协议，这些媒体如电力线、数据（无线和有线）、语音（如，电话）和娱乐媒体。诸如电灯开关和电器设备等大多数家庭控制设备可使用电力线来连接。数据服务可通过宽带（如，DSL 或电缆调制解调器）进入家庭，并可在家庭内使用无线（如，HomeRF 或 802.11B）或有线（如，家庭 PNA、Cat 5、以太网、甚至是电力线）连接来访问。语音话务可通过有线（如，Cat 3）或无线（如，蜂窝电话）进入家庭，并可在家庭中使用 Cat 3 连线来分布。娱乐媒体或其它图形数据可通过卫星或电缆进入家庭，并通常在家庭中使用同轴电缆来分布。IEEE 1394 和 DVI 也是用于媒体设备群集的数字互联。可作为协议标准浮现或已经浮现的所有这些网络环境和其它环境可被互联来形成可通过诸如因特网等广域网连接到外部世界的网络，诸如内联网。简而言之，对数据的存储和传输存在各种不同的源，因此本发明的任何计算设备可按照任何现有方式共享和传输数据，且在本文实施例中描述的方式不旨在是限定性的。

因特网通常指使用传输控制协议/网际协议（TCP/IP）协议套件的网络和网关的集合，该协议在计算机联网领域中是公知的。因特网可被描述为由执行允许用户通过网络交互和共享信息的联网协议的计算机互连的地理上分布的远程计算机网络的系统。由于这类广泛分布的信息共享，诸如因特网等远程网络至今发展成一种开放式系统，开发者可用该开放式系统设计用于执行专用操作或服务的软件应用程序，在本质上没有限制。

由此，网络基础结构启用了诸如客户机/服务器、对等或混合体系结构等大量网络拓朴结构。“客户机”是使用与它无关的另一类或组的服务的一个类或组中的成员。由此，在计算时，客户机是进程，即，粗略地而言是一组请求由另一程序提供的服务的指令或任务。客户机进程利用所请求的服务，而不必“知道”有关其它程序或服务本身的任何工作细节。在客户机/服务器体系结构中，尤其在网络化系统中，客户机通常是访问由例如服务器等另一计算机提供的共享的网络资源的计算机。在图 7A 的图示中，作为一个示例，计算机 920a、920b、920c、920d、920e 等可以被认为是客户机，而计算机 910a、910b 等可以被认为是服务器，其中服务器 910a、910b 等维护随后被复制到客户计算机 920a、920b、920c、920d、920e 等的数据，然而任何计算机都可被认为是客户机、服务器或两者，取决于环境。任何这些计算设备可能正在处理数据或请求服务或任务，这些数据、服务或任务可能隐含本发明的 QFX 的各个实施例的树语法和转换技术。

服务器通常是可通过诸如因特网或无线网络基础架构等远程网络或本地网络访问的远程计算机系统。客户机进程可以在第一计算机系统中活动，而服务器进程可以在第二计算机系统中活动，它们通过通信介质彼此通信，从而提供分布式功能并允许多个客户机利用服务器的信息收集能力。按照本发明的 QFX 利用的任何软件对象可跨多个计算设备或对象分布。

客户机和服务器利用由协议层提供的功能来彼此通信。例如，超文本传输协议（HTTP）是结合万维网（WWW），即“Web”使用的常见协议。通常，诸如网际协议（IP）地址或诸如统一资源定位器（URL）等其它引用的计算机网络地址可以用于彼此标识服务器或客户计算机。网络地址可以被称为 URL 地址。可以通过通信介质来提供通信，例如客户机和服务器可以通过 TCP/IP 连接来彼此耦合以进行大容量通信。

由此，图 9 示出了其中可采用本发明的具有通过网络/总线与客户计算机通信的服务器的示例性联网或分布式环境。更详细而言，根据本发明，多个服务器 910a、910b 等经由通信网络/总线 940 互连，通信网络/总线 14 可以是 LAN、WAN、内联网、GSM 网络、因特网等，它具有多个客户机或远程计算设备 920a、920b、920c、9200d、920e 等，如便携式计算机、手持式计算机、瘦客户机、

联网设备或其它设备，如 VCR、TV、烤箱、灯、加热器等等。因此构想了，本发明可应用于期望根据本发明定义的 Q 框架编译和执行属性化树语法并转换成 MFST 的任何计算设备。

例如，在其中通信网络/总线 940 是因特网的网络环境中，服务器 910a、910b 等可以是客户机 920a、920b、920c、920d、920e 等通过诸如 HTTP 等多种已知协议中的任一种与其通信的 web 服务器。服务器 910a、910b 等也可担当客户机 920a、920b、920c、920d、920e 等，这是分布式计算环境的特性。

如上所述，通信可以是有线或无线的，或者在适当时是两者的组合。客户机设备 920a、920b、920c、920d、920e 等可以通过或不通过通信网络/总线 14 通信，并可具有与其相关联的独立通信。例如，在 TV 或 VCR 的情况下，可以有或没有其控制的网络化方面。每一客户计算机 920a、920b、920c、920d、920e 等以及服务器计算机 910a、910b 等可以具备各种应用程序模块或对象 135a、135b、135c 等，并具有对各种类型的存储元件或对象的连接或访问，在这些存储元件或对象上可储存文件或数据流，或者可向其下载、发送或迁移文件或数据流的各部分。计算机 910a、910b、920a、920b、920c、920d、920e 等中的任何一个或多个可负责维护并更新数据库 930 或其它存储元件，诸如用于储存根据本发明处理或保存的数据的数据库或存储器 930。由此，本发明可以用于具有可访问计算机网络/总线 940 并与其交互的客户计算机 920a、920b、920c、920d、920e 等，以及可与客户机计算机 920a、920b、920c、920d、920e 等交互的服务器计算机 910a、910b 等，以及其它类似的设备和数据库 930 的计算机网络环境中。

示例性计算设备

如本文所述，本发明适用于可能期望应用根据本发明定义的 QFX 技术的任何设备。从而，应当理解，构想了所有种类的手持式、便携式和其它计算设备和计算对象来用于本发明，即，在设备可实现表示状态机的软件进程或以其它方式接收、处理或存储数据的任何地方。因此，在下面的图 10 中描述的以下通用远程计算机仅是一个示例，且本发明可用具有网络/总线互操作性和交互的任何客户机来实现。由此，本发明可在其中蕴含了极少或最小客户机资源的联网的主存服务的环境，例如其中客户机设备仅用作到网络/总线的接口－如置

于电器中的对象－的联网环境中实现。

尽管并非所需，但本发明可以部分地经由操作系统来实现，以供设备或对象的服务开发者使用，和/或被包括在结合本发明的组件操作的应用软件中。软件可以在由诸如客户机工作站、服务器或其它设备等一个或多个计算机执行的诸如程序模块等计算机可执行指令的通用上下文中描述。本领域的技术人员可以理解，本发明可以用其它计算机系统配置和协议来实施。

图 10 由此示出了其中可实现本发明的合适的计算系统环境 1000a 的一个示例，但如以上清楚地描述的，计算系统环境 1000a 仅为用于介质设备的合适的计算环境的一个示例，并非对本发明的使用范围或功能提出任何局限。也不应将计算系统 1000a 解释为对示例性操作环境 1000a 中示出的任何组件或其组合具有任何依赖性或需求。

参见图 10，用于实现本发明的示例性远程设备包括计算机 1010a 形式的通用计算设备。计算机 1010a 的组件可以包括，但不限于，处理单元 1020a、系统存储器 1030a 和将包括系统存储器在内的各种系统组件耦合至处理单元 1020a 的系统总线 1021a。系统总线 1021a 可以是几种类型的总线结构中的任一种，包括存储器总线或存储控制器、外围总线、以及使用各种总线体系结构中的任一种的局部总线。

计算机 1010a 通常包括各种计算机可读介质。计算机可读介质可以是可由计算机 1010a 访问的任何可用介质。作为示例而非局限，计算机可读介质可以包括计算机存储介质和通信介质。计算机存储介质包括以用于存储诸如计算机可读指令、数据结构、程序模块或其它数据等信息的任何方法或技术实现的易失性和非易失性、可移动和不可移动介质。计算机存储介质包括但不限于，RAM、ROM、EEPROM、闪存或其它存储器技术、CDROM、数字多功能盘（DVD）或其它光盘存储、磁盒、磁带、磁盘存储或其它磁存储设备、或可以用来储存所期望的信息并可由计算机 1,010a 访问的任何其它介质。通信介质通常以诸如载波或其它传输机制等已调制数据信号来体现计算机可读指令、数据结构、程序模块或其它数据，并包括任意信息传送介质。

系统存储器 1030a 可以包括诸如只读存储器(ROM)和/或随机存取存储器(RAM)等易失性和/或非易失性存储器形式的计算机存储介质。基本输入/输出

系统（BIOS）可被存储在存储器 1030a 中，它包含帮助在诸如启动期间在计算机 1010a 内元件之间传递信息的基本例程。存储器 1030a 通常还包含处理单元 1020a 可以立即访问和/或目前正操作的数据和/或程序模块。作为示例而非局限，存储器 1030a 还可以包括操作系统、应用程序、其它程序模块、和程序数据。

计算机 1010a 也可以包括其它可移动/不可移动、易失性/非易失性计算机存储介质。例如，计算机 1010a 可以包括对不可移动、非易失性磁介质进行读写的硬盘驱动器，对可移动、非易失性磁盘进行读写的磁盘驱动器，和/或对诸如 CD-ROM 或其它光学介质等可移动、非易失性光盘进行读写的光盘驱动器。可以在示例性操作环境中使用的其它可移动/不可移动、易失性/非易失性计算机存储介质包括但不限于，磁带盒、闪存卡、数字多功能盘、数字录像带、固态 RAM、固态 ROM 等等。硬盘驱动器通常由诸如接口等不可移动存储器接口连接至系统总线 1021a，而磁盘驱动器或光盘驱动器通常由诸如接口等可移动存储器接口连接至系统总线 1021a。

用户可以通过输入设备，如键盘和定点设备（通常指鼠标、跟踪球或触摸板）向计算机 1010a 输入命令和信息。其他输入设备可以包括话筒、操纵杆、游戏手柄、圆盘式卫星天线、扫描仪等等。这些和其它输入设备通常由耦合至系统总线 1021a 的用户输入 1040a 和相关联的接口连接到处理单元 1020a，但是也可由诸如并行端口、游戏端口或通用串行总线（USB）之类的其它接口和总线结构连接。图形子系统也可以被连接到系统总线 1021a。监视器或其它类型的显示设备也通过接口，如输出接口 1050a 连接至系统总线 1021a，而输出接口 1050a 又与视频存储器通信。除监视器之外，计算机还可以包括其它外围输出设备，如扬声器和打印机，它们可以通过输出接口 1050a 连接。

计算机 1010a 可使用至诸如远程计算机 1070a 等的一个或多个远程计算机的逻辑连接在网络化或分布式环境中操作，远程计算机 1070a 又可以具有与设备 1010a 不相同的媒体能力。远程计算机 1070a 可以是个人计算机、服务器、路由器、网络 PC、对等设备或其它常见的网络节点、或任何其它远程媒体消费或传输设备，并且可以包括上面关于计算机 1010a 所描述的任何或全部元件。图 10 所示的逻辑连接包括诸如局域网（LAN）或广域网（WAN）等的网络 1071a，

但也可以包括其它网络/总线。这样的联网环境在家庭、办公室、企业范围计算机网络、内联网和因特网中是常见的。

当在 LAN 联网环境中使用时，计算机 1010a 通过网络接口或适配器连接至 LAN 1071a。当在 WAN 联网环境中使用时，计算机 1010a 通常包括通信组件，诸如调制解调器或用于通过诸如因特网等的 WAN 建立通信的其它装置。诸如调制解调器等通信组件可以是内置或外置的，它可以通过输入 1040a 的用户输入接口或其它适当的机制连接至系统总线 1021a。在网络化环境中，相对于计算机 1010a 所描述的程序模块或其部分可被储存在远程存储器存储设备中。可以理解，所示和所述的网络连接是示例性的，且可以使用在计算机之间建立通信链路的其它手段。

示例性分布计算体系结构

鉴于个人计算和因特网的交汇，已经开发且正在开发各种分布式计算框架。个人和商业用户同样地拥有用于应用程序和计算设备的无缝的互操作和启用 web 的接口，使得计算活动越来越面向 web 浏览器和网络。

例如，MICROSOFT®的托管代码平台，即.NET 包括服务器、诸如基于 web 的数据存储等构件块服务、以及可下载设备软件。一般而言，.NET 平台提供（1）令整个范围的计算设备共同工作并在所有设备上自动更新并同步用户信息的能力，（2）提高的网页交互能力，通过大量使用 XML 而不是 HTML 来实现，（3）从用于各种应用，如电子邮件，或软件，如 Office.NET 的管理的中央起点到用户的具有产品和服务的定制访问和传送的特点的在线服务，（4）中央化数据存储，将增加对信息访问以及用户和设备间的信息同步的效率和简易性，（5）集成各种通信介质，如电子邮件、传真和电话的能力，（6）对开发员来说，创建可重复使用模块的能力，借此提高生产力并降低编程错误数，以及（7）还有许多其它跨平台和语言综合特性。

尽管此处的某些示例性实施例是结合诸如应用程序编程接口（API）等驻留在计算设备上的软件来描述的，但本发明的一个或多个部分也可以通过操作系统、“中间人”对象、控制对象、硬件、固件、中间语言指令或对象等来实现，使根据本发明的 QFX 的实施例可以被包括在由诸如.NET 代码等托管代码启用的所有语言和服务中，以及在其它分布式计算框架中，在其中得到支持

或经由它们来访问。

有多种实现本发明的方法，例如适当的 API、工具箱、驱动程序代码、操作系统、控件、独立或可下载软件对象等，它们使得能够使用本发明的 QFX。本发明从 API（或其它软件对象）的观点以及从可实现根据本发明的 QFX 的至少一部分的数据结构、软件或硬件对象的观点构想了对本发明的使用。由此，此处描述的本发明的各种实现都可以具有完全采用硬件、部分采用硬件且部分采用软件、以及采用软件的方面。

在此使用的词语“示例性”意味着用作示例、实例或说明。为避免疑问，本文公开的主体不受限于这样的示例。此外，本文描述为“示例性”的任何方面或设计不必解释成优于其它方面或设计或比其它方面或设计有利，它也不旨在排除本领域的普通技术人员所知的等效示例性结构和技术。而且，就术语“包括”、“具有”、“包含”和其它类似的词语在详细描述或权利要求书中使用而言，为避免疑问，这样的术语旨在以类似于术语“包括”作为开放的过渡词的方式解释而不排除任何附加或其它元素。

如上所述，尽管结合各种计算设备和网络体系结构描述了本发明的示例性实施例，但基本概念可被应用于其中期望包括 MFST 实现变换的任何计算设备或系统。例如，本发明的算法和硬件实现可被应用于计算设备的操作系统，可作为设备上的独立对象、作为另一对象的一部分、作为可重复使用的控件、作为可从服务器下载的对象、作为设备或对象和网络之间的“中间人”、作为分布式对象、作为硬件、以存储器、以上任何的组合等来提供。尽管此处选择了示例性编程语言、名称和示例来表示各种选择，但这些语言、名称和示例不旨在为限制性的。本领域的普通技术人员将认识到，有多种方法来提供实现本发明的各实施例所实现的相同、相似或等效的功能的目标代码和命名法。

如上所述，此处所述的各种技术可结合硬件或软件，或在适当时以两者的组合来实现。如在此所使用地，术语“组件”、“系统”等同样指的是计算机相关实体，或者是硬件、硬件和软件的组合、软件或执行中的软件。例如，组件可以是但不限于是，在处理器上运行的进程、处理器、对象、可执行码、执行的线程、程序和/或计算机。作为说明，运行在计算机上的应用程序和计算机本身都可以是计算机组件。一个或多个组件可以驻留在进程和/或执行的线程

内，并且组件可以位于一个计算机上和/或分布在两个或更多的计算机之间。

由此，本发明的方法和装置或其特定方面或部分可采取包含在诸如软盘、CD-ROM、硬盘驱动器或任何其它机器可读存储介质等有形介质中的程序代码（即，指令）的形式，其中当程序代码被加载到诸如计算机等机器内并由其执行时，该机器成为用于实现本发明的装置。在程序代码在可编程计算机上执行的情况下，计算设备通常包括处理器、该处理器可读的存储介质（包括易失性和非易失性的存储器和/或存储元件）、至少一个输入设备、以及至少一个输出设备。可例如通过使用数据处理 API、软件对象等来实现或利用本发明的 QFX 的一个或多个程序较佳地用高级过程语言或面向对象的编程语言来实现以与计算机系统通信。然而，如果需要，程序可以用汇编语言或机器语言来实现。在任何情形中，语言可以是编译的或解释的语言，且与硬件实现相结合。

本发明的方法和装置也可以经由以通过某种传输介质传输的程序代码的形式体现的通信来实现，比如通过电线或电缆、通过光纤或经由任何其它传输形式，其中，当程序代码由诸如 EPROM、门阵列、可编程逻辑器件（PLD）、客户计算机等机器接收、加载并执行时，该机器成为用于实现本发明的装置。当在通用处理器上实现时，程序代码与处理器相结合来提供一种用于调用本发明的功能的独特装置。另外，结合本发明使用的任何存储技术总是可以是硬件和软件的组合。

此外，所公开的主题可以使用产生软件、固件、硬件或其任意组合的标准编程和/或工程技术实现为用于控制基于计算机或处理器的设备以实现在此所详述的诸方面的系统、方法、装置或制品。此处所用的术语“制品”（或作为替换，“计算机程序产品”）旨在涵盖可从任何计算机可读设备、载体或介质访问的计算机程序。例如，计算机可读介质可以包括但不限于磁存储设备（例如，硬盘、软盘、磁带……）、光盘（例如，压缩盘（CD）、数字通用盘（DVD）……）、智能卡和闪存设备（例如，卡、棒）。另外知道，可以采用载波来承载计算机可读电子数据，例如那些用于发送和接收电子邮件或用于访问如因特网或局域网（LAN）等网络的数据。

已经关于若干组件之间的交互描述了前述系统。应该理解，这样的系统和组件可以包括那些组件或子组件，所指定组件或子组件中的一些和/或另外的组

件，并根据前述的各种排列和组合。子组件也可以被实现为通信耦合至其它组件而非被包括在父组件（分层）内的组件。另外，应注意到一个或多个组合可被组合成提供聚集功能的单个组件，或被分成若干单独的子组件，且诸如管理层的任何一个或多个中间层可被设置成通信耦合到这样的子组件以便提供集成功能。此处描述的任何组件也可以与在此未具体描述但本领域的技术人员公知的一个或多个其他组件交互。

考虑到以上描述的示例性系统，参考图 5 的流程图将可以更好地理解依照所公开的主题实现的方法。尽管出于说明简单的目的，各方法被显示和描述为一系列框，但应该理解和领会，所要求保护的主题不受框次序的限制，因为一些框能够以不同的次序和/或与在此描绘和描述的其它框同时发生，或以其它方式提供等效的功能。尽管经由流程图示出了非顺序或分支的流程，但可以理解，可实现达成相同或类似结果的各种其它分支、流程路径和块次序。而且，并非所有示出的框都是实现以下描述的方法所必需的。

此外，应该明白以上公开的系统以及以下方法的不同部分可以包括或包含人工智能或基于知识或规则的组件、子组件、进程、装置、方法或机制（例如，支持向量机、神经网络、专家系统、贝叶斯置信网络、模糊逻辑、数据融合引擎、分类器等）。这样的组件和其它组件可以自动化执行某些机制或进程，由此使得 QFX 的各部分更为自适应、高效及智能。

尽管已结合各个附图的优选实施方式对本发明进行了描述，但是可以理解，可以使用其它类似的实施方式，或可以对所述实施方式进行修改或添加，来实现本发明的相同功能而不背离本发明。例如，在诸如对等联网环境等联网环境的上下文中描述了本发明的示例性网络环境，但是本领域的技术人员将认识到，本发明不限于此，并且本申请中所描述的方法可应用于任何计算设备或环境，诸如游戏控制台、手持式计算机、便携式计算机等等，不论其是有线还是无线的，并且该方法可应用于经由通信网络连接并通过网络交互的任意数量的此类计算设备。此外，应当强调，构想了包括手持式设备操作系统和其它应用专用操作系统的各种计算机平台，尤其是在无线联网设备的数量持续增长时。

尽管各示例性实施例涉及在特定编程语言构造的上下文中利用本发明，但

是本发明不限于此，而是可用任何语言实现并用于转换包括动作的任何语法。而且，本发明可以在多个处理芯片或设备中实现或跨多个处理芯片或设备实现，且存储可以类似地跨多个设备来实现。因此，本发明不应限于任何单个实施例，而是应该根据所附权利要求书的广度和范围来解释。

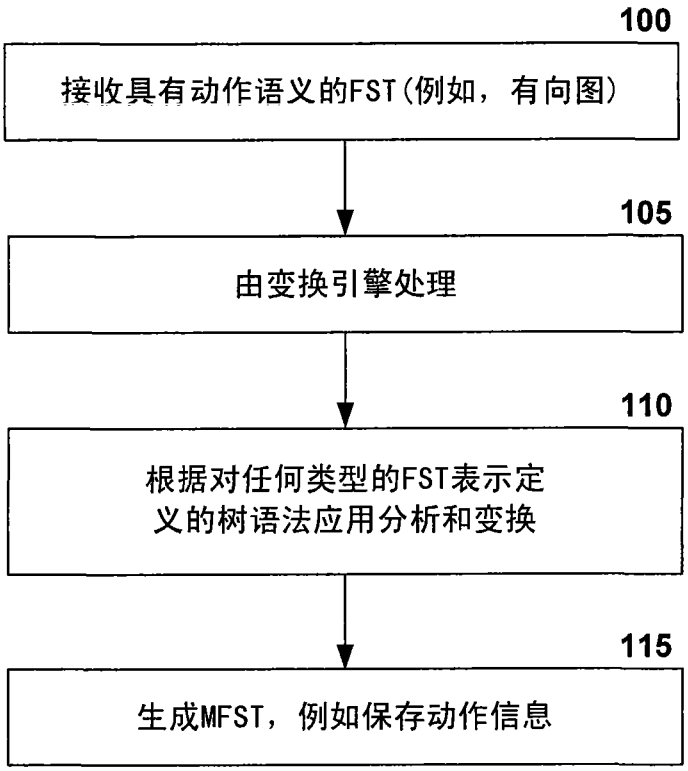

100
接收具有动作语义的FST(例如，有向图)
105
由变换引擎处理
110
根据对任何类型的FST表示定义的树语法应用分析和变换
115
生成MFST，例如保存动作信息

图 1A

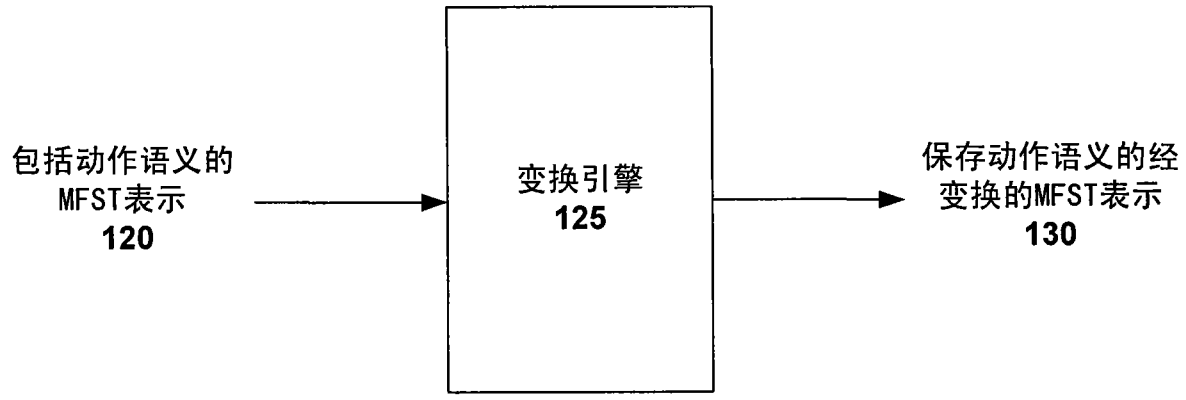

包括动作语义的MFST表示
120
变换引擎
125
保存动作语义的经变换的MFST表示
130

图 1B

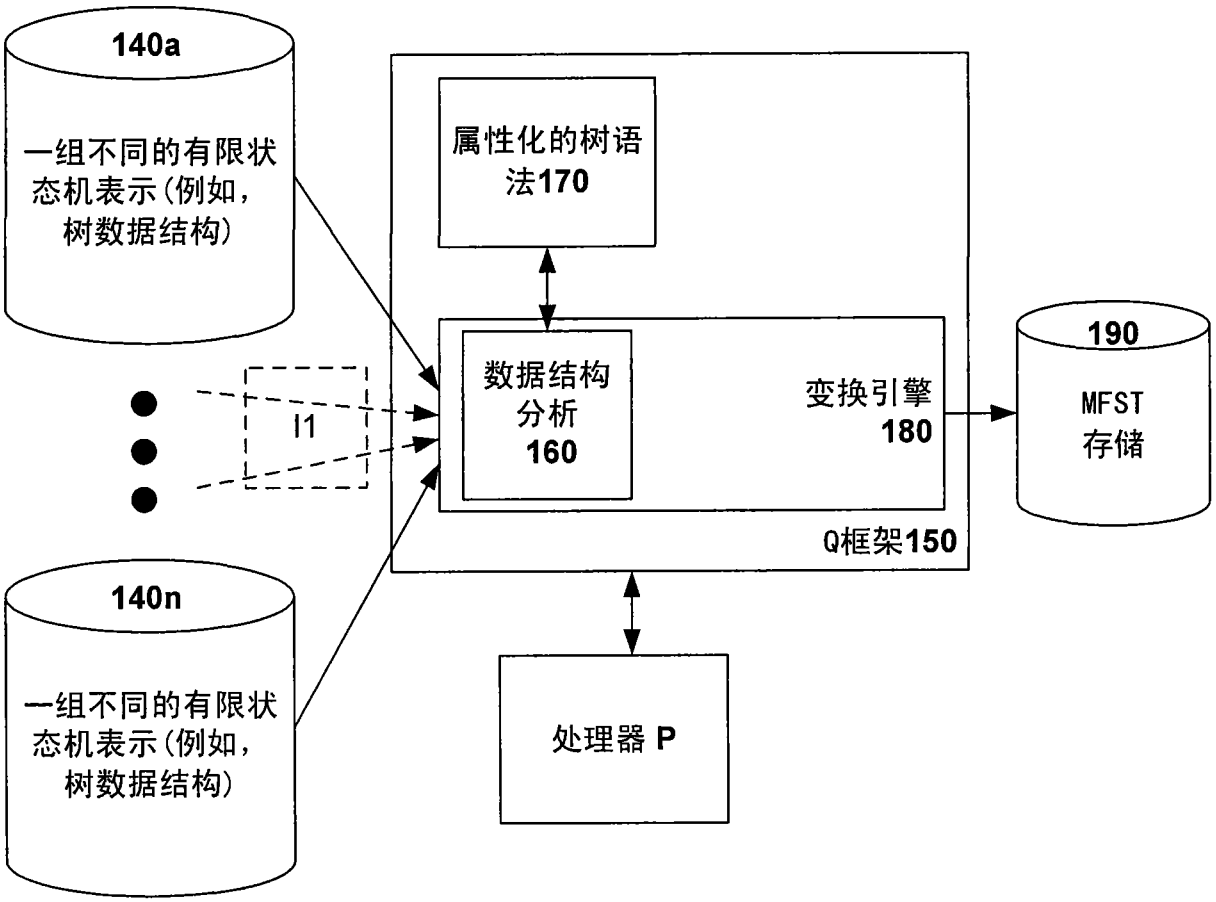

140a
一组不同的有限状态机表示(例如，树数据结构)
属性化的树语法170
数据结构分析160
变换引擎180
190
MFST存储
I1
Q框架150
140n
一组不同的有限状态机表示(例如，树数据结构)
处理器 P

图 1C

```
interface XEControlInstructions {
        IEnvironment CurrentEnvironment { get; set; }
        IContinuation CurrentContinuation { get; }
        IContinuation Call(XEInstance target, IContinuation continuation,
                        IEnvironment callingEnvironment);
        IContinuation Return();
        void Mark();
        object Yield();
        IEnvironment NewEnvironment();
        void PushEnvironment(string variableName);
        void PopEnvironment();
        void Exec(ActionReference action);
}
```

200

图 2A

```
void PushEnvironment(string variableName) {
        IEnvironment tempEnv;
        environmentStack.Push(CurrentEnvironment);
        tempEnv=NewEnvironment();
        Bind(varName,tempEnv);
        CurrentEnvironment=tempEnv;
        Mark();
}
```

210

图 2B

```
220 void PopEnvironment() {
        Bind(''term'',Yield());
        environmentStack.Pop();
        CurrentEnvironment=environmentStack.Peek();
    }
```

图 2C

```
230 interface XEEnvironmentInstructions {
        IEnvironment ChildEnv();
        bool Bind(string variableName,object value);
        bool Lookup(string variableName,out object value);
    }
```

图 2D

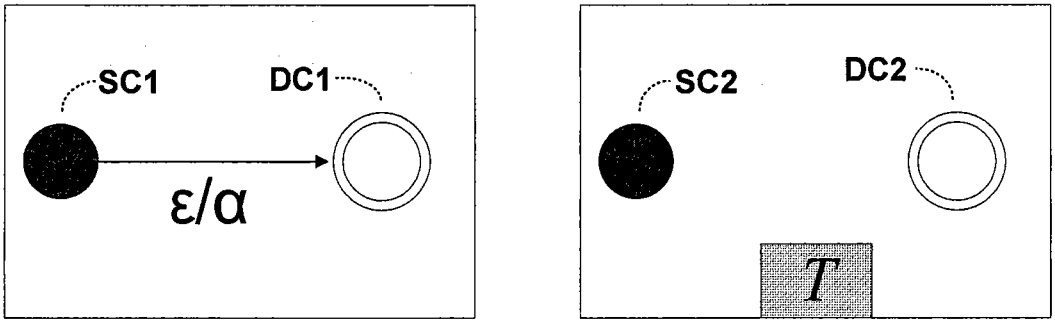
```
public static Set<State<T>> EClosure(Set<State<T>> nfaStates,
                                     List<Action> actions) {
        Set<State<T>> eClosure = new Set<State<T>>();
        Stack<State<T>> stack = new Stack<State<T>>();

        foreach (State<T> nfaState in nfaStates) {
                stack.Push(nfaState);
                eClosure.Insert(nfaState);
        }

        while (stack.Count > 0) {
                State<T> nfaSrc = stack.Pop();
240             foreach (Transition<T> t in nfaSrc.EpsTransitions) {
(A1)                    if (t.Action!=null)
(A2)                            actions.Add(t.Action);
                        foreach (State<T> nfaDst in t.States) {
                                if (!(eClosure.Contains(nfaDst))) {
                                        eClosure.Insert(nfaDst);
                                        stack.Push(nfaDst);
                                }
                        }
                }
        }
        return (eClosure);
}
```

图 2E

```
public NFA<T> SubsetConstruction(List<T> lexicon) {
        State<T> startDFAState =
                State<T>.SubsetState(State<T>.EClosure(startState));
        NFA<T> dfa = SubsetDFA(startDFAState);

        Queue<State<T>> q = new Queue<State<T>>();
        q.Enqueue(startDFAState);

        while (q.Count > 0) {
                State<T> src = q.Dequeue();
                foreach (T symbol in lexicon) {
(B1)                    List<Actions> symActions=new List<Actions>();
(B2)                    Set<State<T>> neighbors=src.Move(symbol,symActions);
(B3)                    List<Actions> epsActions=new List<Actions>();
(B4)                    Set<State<T>> eclosure=
                                State<T>.EClosure(neighbors,epsActions);

                        State<T> dst = State<T>.SubsetState(eclosure);
(B5)                    src.AddTransition(symbol,dst,symActions);
(B6)                    src.AddActions(epsActions);
                        return (dfa);
                }
        }
}
```

250

图 2F


SC1
DC1
ε/α
SC2
DC2
T

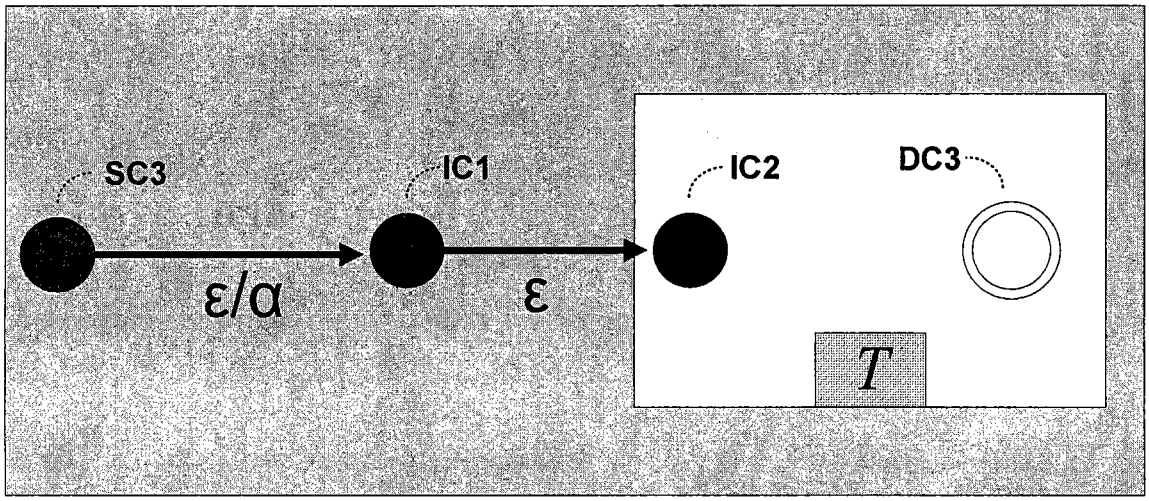

SC3
IC1
ε/α
ε
IC2
DC3
T

图 3A

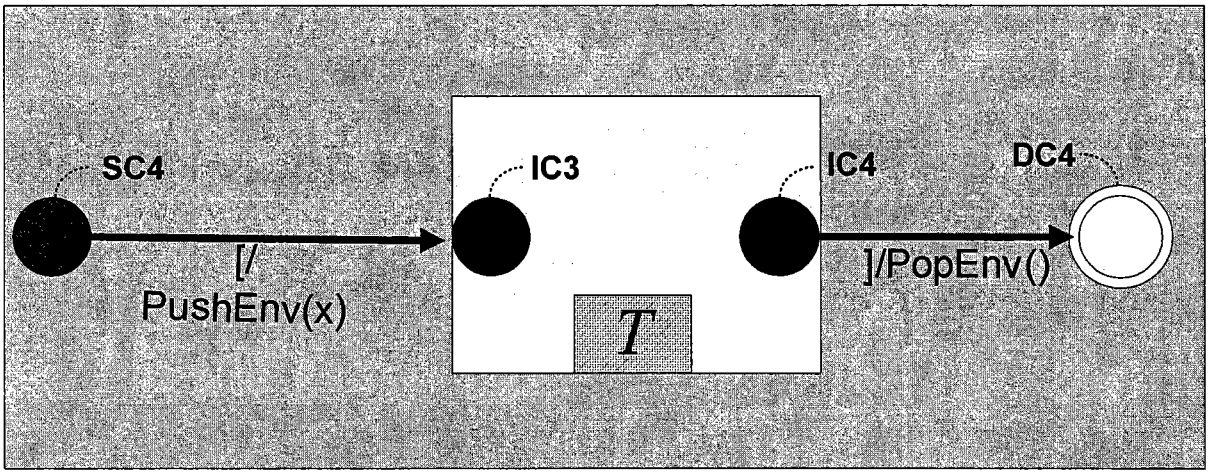

SC4
[/
PushEnv(x)
IC3
T
IC4
]/PopEnv()
DC4

图 3B

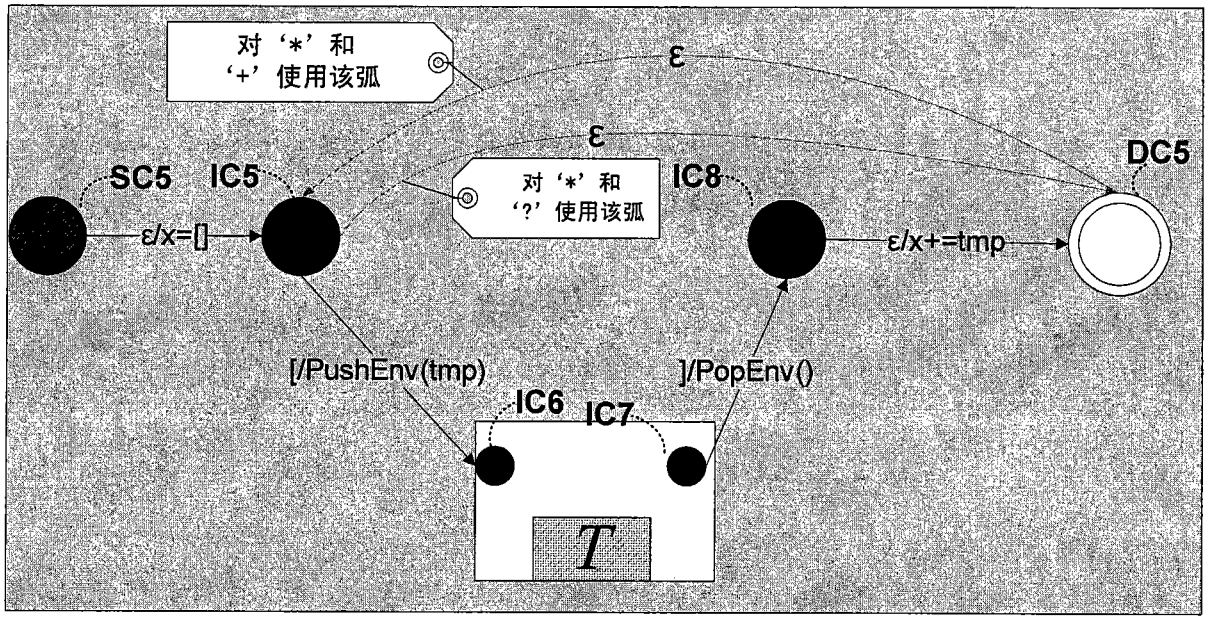

对‘*’和
‘+’使用该弧
ε
ε
对‘*’和
‘?’使用该弧
SC5
IC5
IC8
DC5
ε/x=[]
ε/x+=tmp
[/PushEnv(tmp)
]/PopEnv()
IC6
IC7
T

图 3C

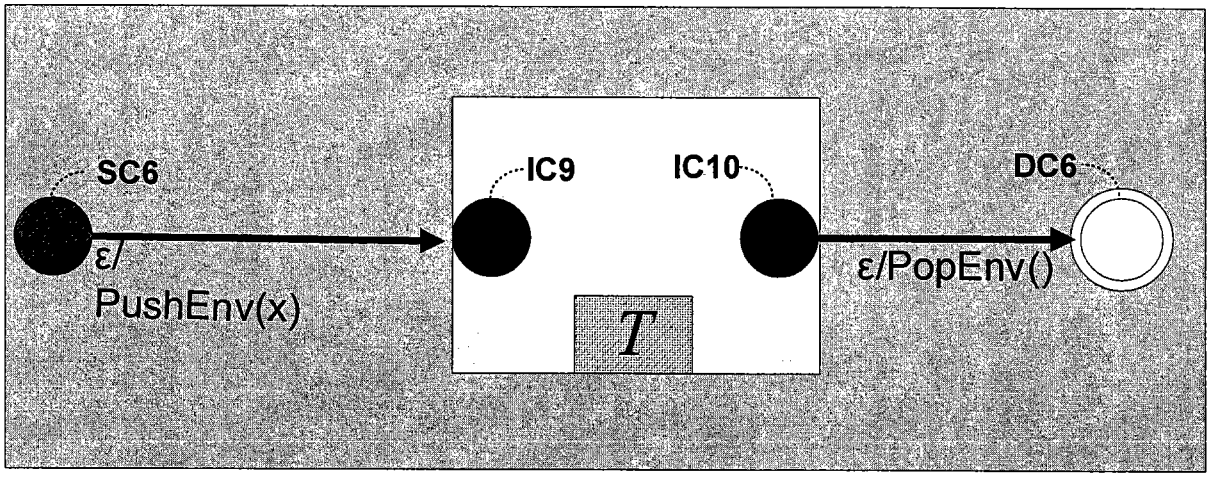

SC6
IC9
IC10
DC6
ε/
PushEnv(x)
ε/PopEnv()
T

图 3D

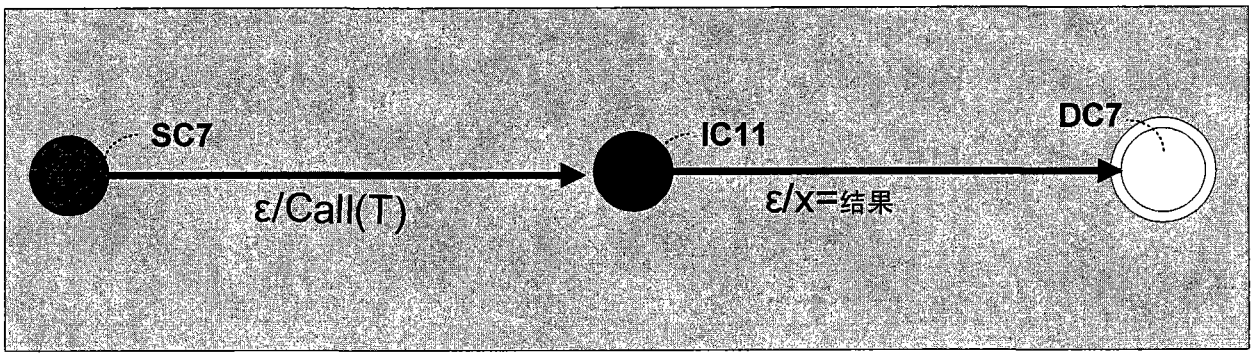

SC7
ε/Call(T)
IC11
ε/X=结果
DC7

图 3E

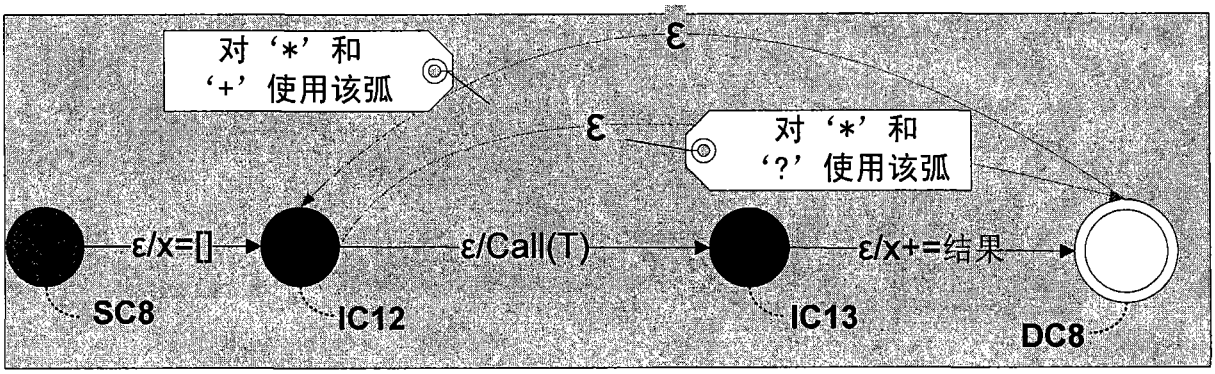

对‘*’和
‘+’使用该弧
ε
ε
对‘*’和
‘?’使用该弧
ε/X=[]
ε/Call(T)
ε/X+=结果
SC8
IC12
IC13
DC8

图 3F

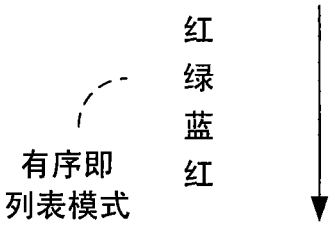


图 4A

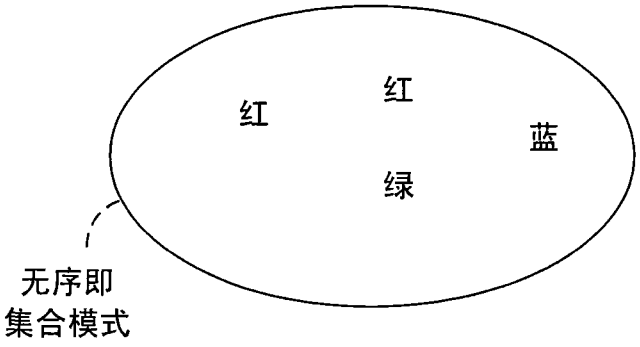


图 4B

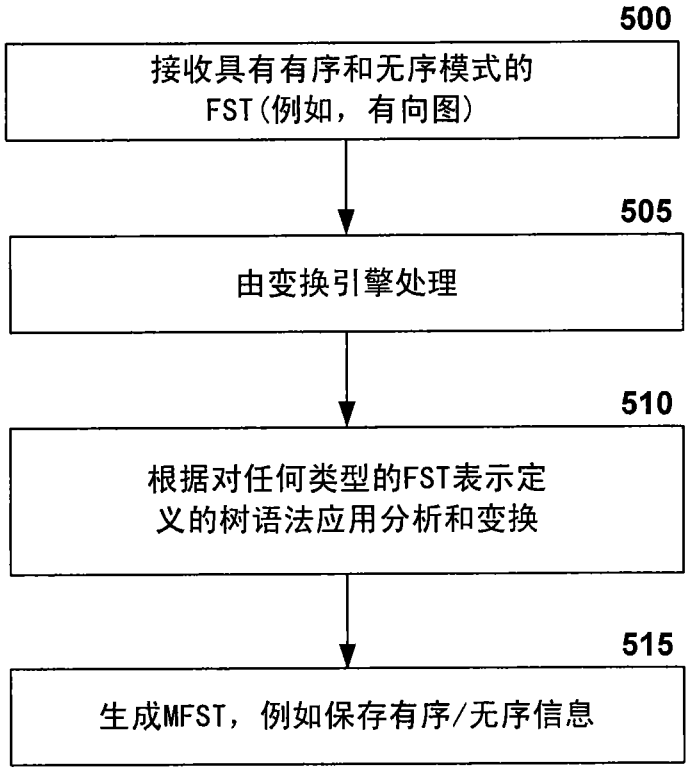

500
接收具有有序和无序模式的FST(例如，有向图)
505
由变换引擎处理
510
根据对任何类型的FST表示定义的树语法应用分析和变换
515
生成MFST，例如保存有序/无序信息

图 5A

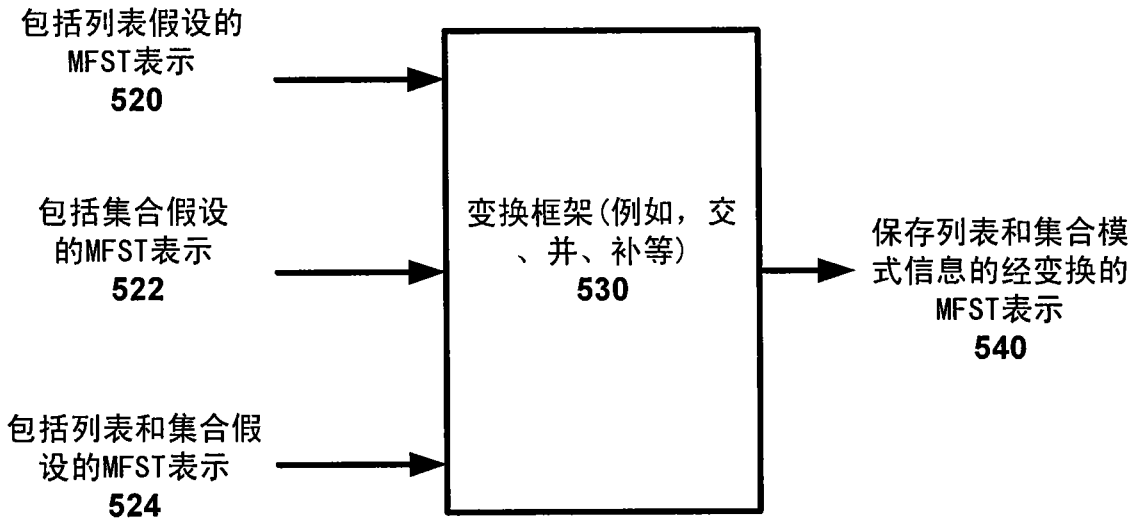

包括列表假设的MFST表示
520
包括集合假设的MFST表示
522
包括列表和集合假设的MFST表示
524
变换框架(例如，交、并、补等)
530
保存列表和集合模式信息的经变换的MFST表示
540

图 5B

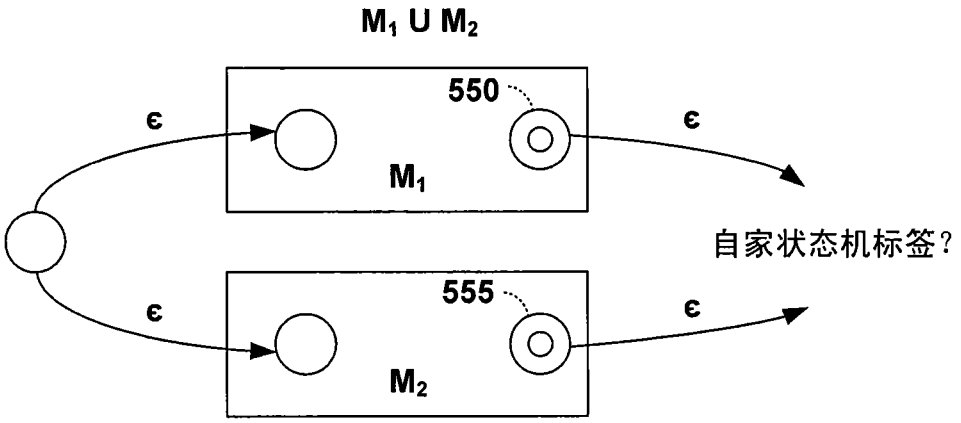

M1 U M2
550
ϵ
ϵ
M1
自家状态机标签?
555
ϵ
ϵ
M2

图 5C

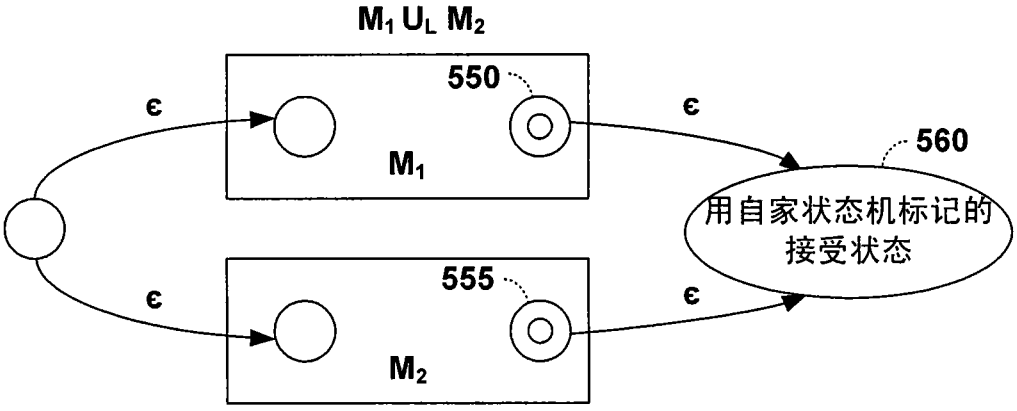

M1 UL M2
550
ϵ
ϵ
M1
560
用自家状态机标记的
接受状态
555
ϵ
ϵ
M2

图 5D

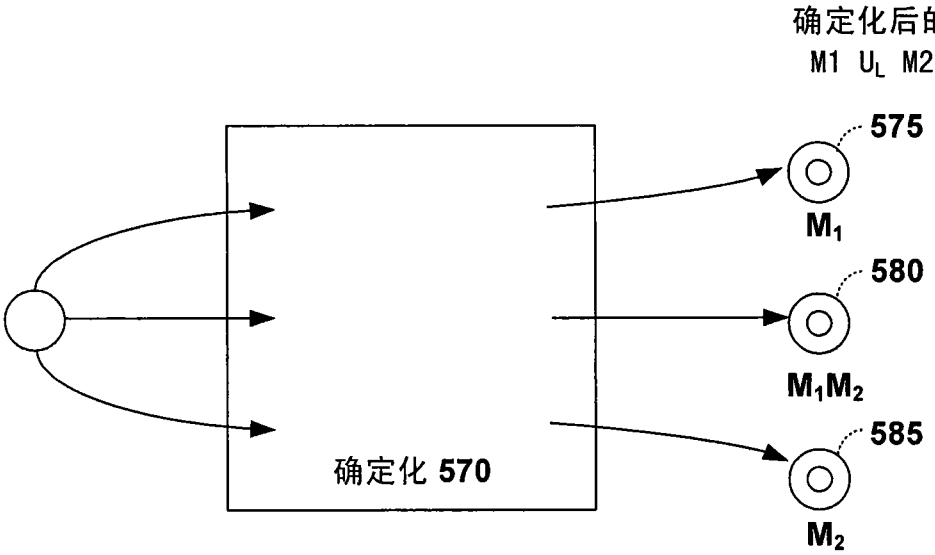

确定化后的
M1 UL M2
575
M1
580
M1M2
585
M2
确定化 570

图 5E

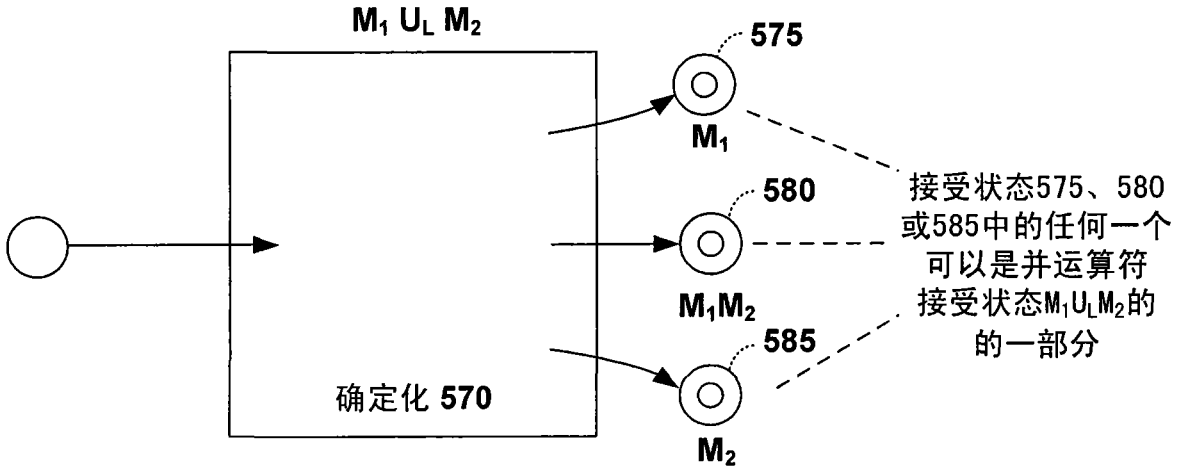

M1 UL M2
575
M1
580
M1M2
585
M2
确定化 570
接受状态575、580
或585中的任何一个
可以是并运算符
接受状态M1ULM2的
的一部分

图 5F

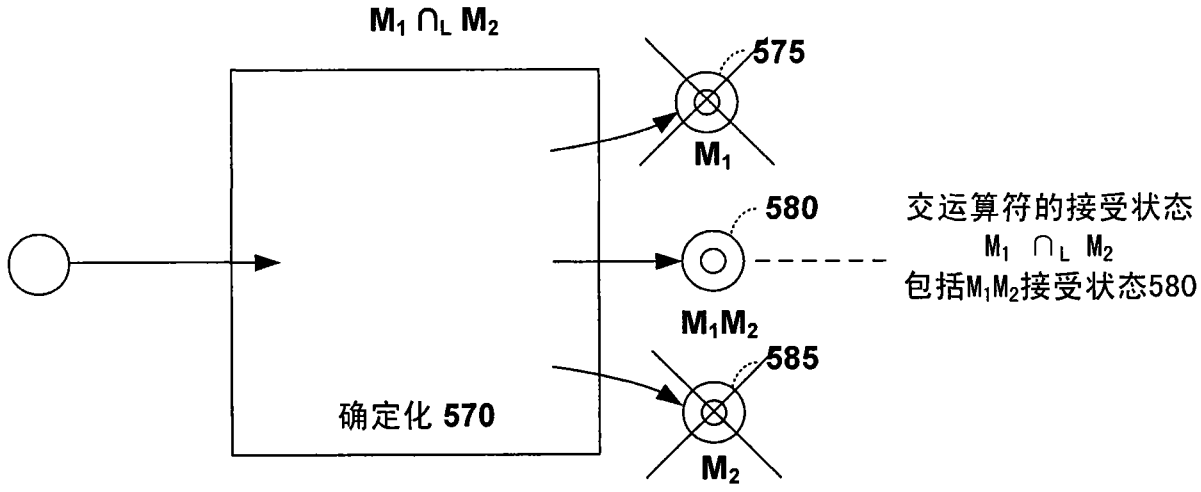

M1 ∩L M2
575
M1
580
M1M2
585
M2
确定化 570
交运算符的接受状态
M1 ∩L M2
包括M1M2接受状态580

图 5G

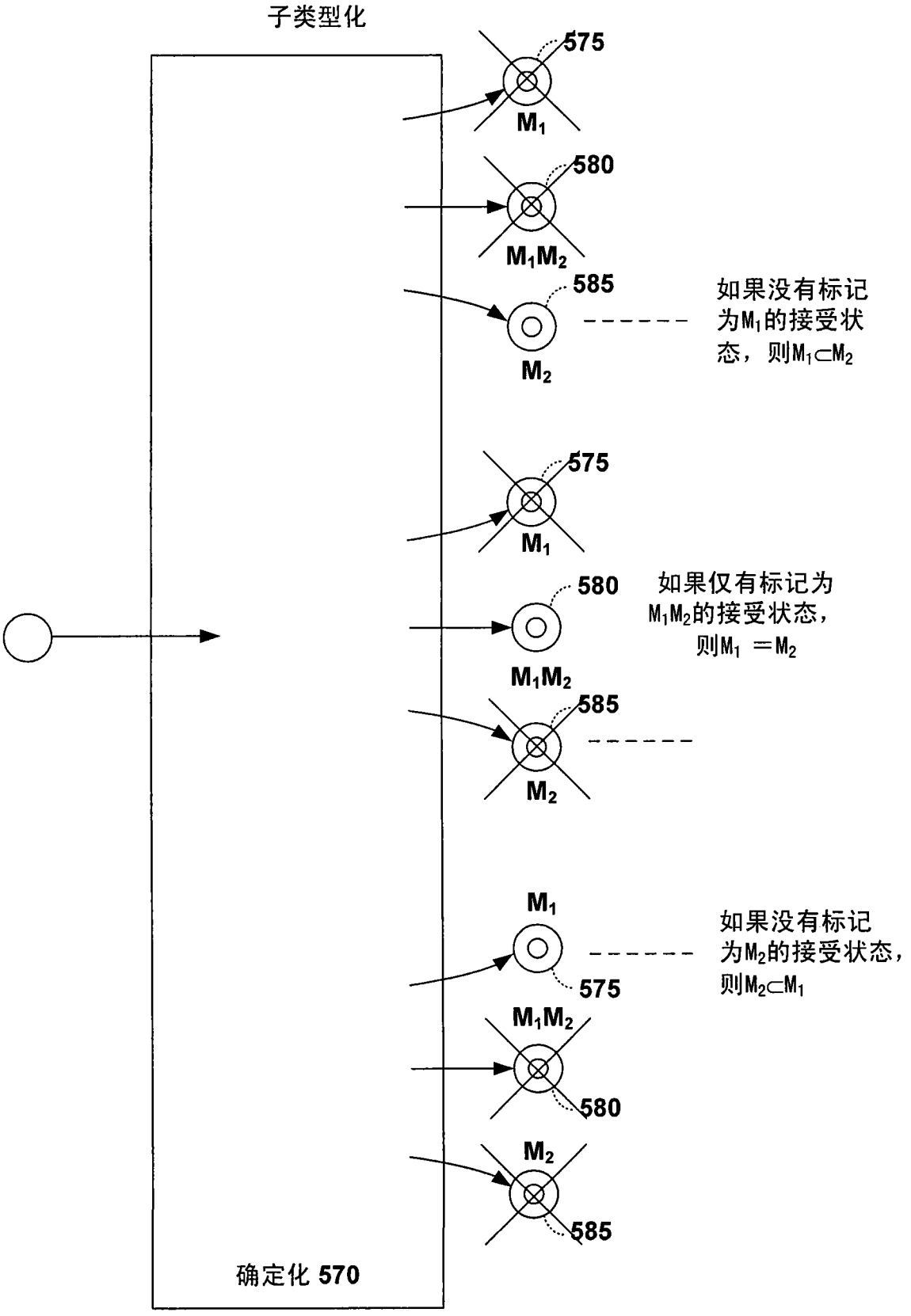

子类型化
575
M1
580
M1M2
585
M2
如果没有标记为M1的接受状态，则M1⊂M2
575
M1
580
M1M2
如果仅有标记为M1M2的接受状态，则M1 =M2
585
M2
M1
575
如果没有标记为M2的接受状态，则M2⊂M1
M1M2
580
M2
585
确定化 570

图 5H

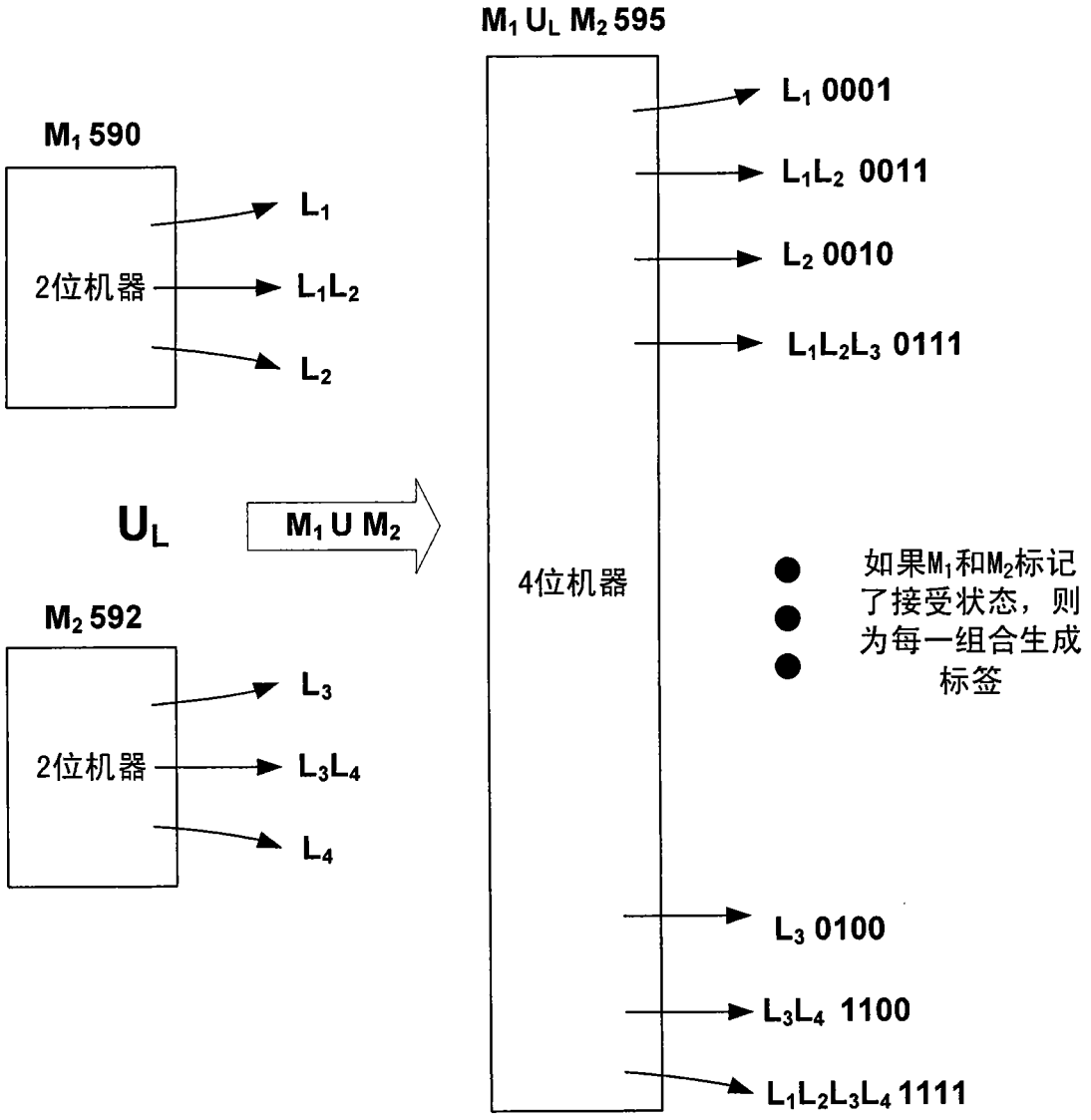

M1 UL M2 595
M1 590
2位机器
L1
L1L2
L2
UL
M1 U M2
M2 592
2位机器
L3
L3L4
L4
4位机器
L1 0001
L1L2 0011
L2 0010
L1L2L3 0111
如果M1和M2标记了接受状态，则为每一组合生成标签
L3 0100
L3L4 1100
L1L2L3L4 1111

图 5I

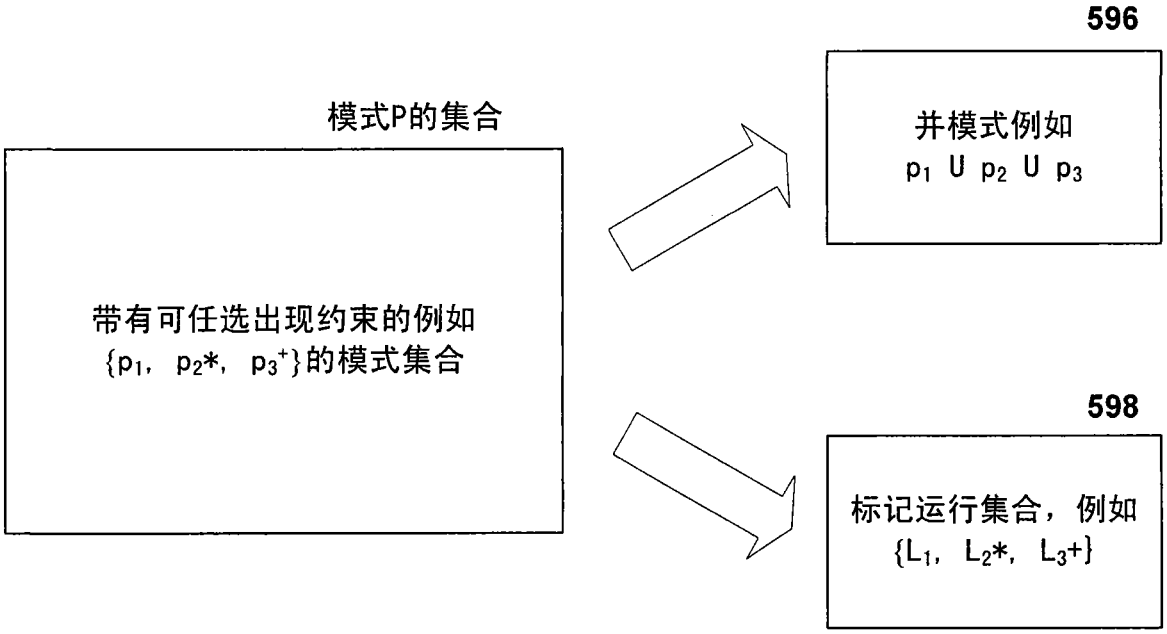
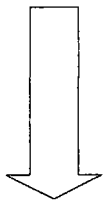
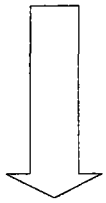

596
模式P的集合
并模式例如
p_1 U p_2 U p_3
带有可任选出现约束的例如
{p_1, p_2*, p_3^+}的模式集合
598
标记运行集合，例如
{L_1, L_2*, L_3+}

图 5J

$$P_1 \cup_L P_2$$



$$\cup(P_1) \quad \cup(P_2) \quad \Gamma(P_1) \quad \Gamma(P_2)$$



$$\cup(P_1 \cup_L P_2) = \cup(P_1) \cup_L \cup(P_2)$$

$$\&$$

$$\Gamma(P_1 \cup_L P_2) = \Gamma(P_1) * \Gamma(P_2)$$

标记并叉积LUCP

图 5K

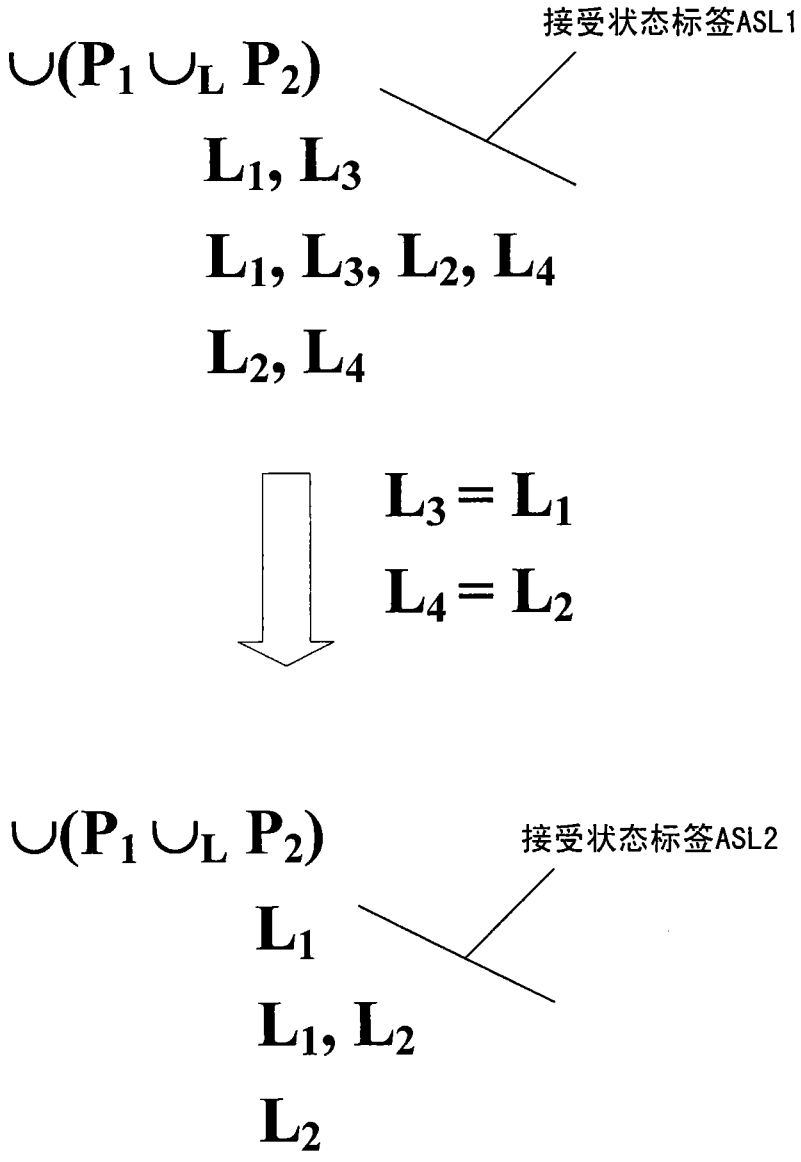

接受状态标签ASL1
∪(P1 ∪L P2)
L1, L3
L1, L3, L2, L4
L2, L4
L3 = L1
L4 = L2
∪(P1 ∪L P2)
接受状态标签ASL2
L1
L1, L2
L2

图 5L

图 6A

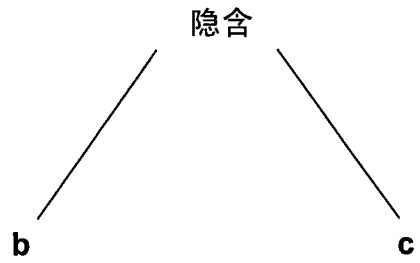


图 6B

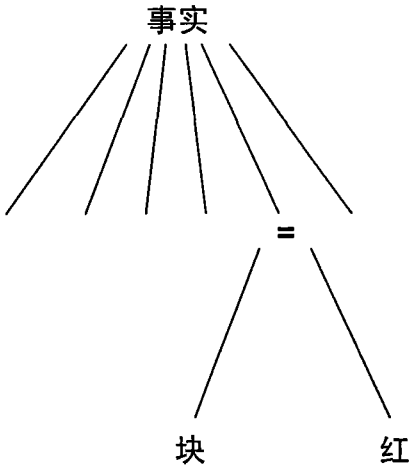


图 6C

状态转移表STT1

当前状态/条件	状态 A	状态 B	状态 C
条件 X	...	...	...
条件 Y	...	状态C	...
条件 Z	...	...	...

图 7A

状态表ST1

状态名SN	条件CO	动作AC
当前状态	进入动作	输出名
	退出动作	输出名
	虚拟条件	输出名
	...	...
下一状态名	虚拟条件	输出名
下一状态名	虚拟条件	输出名
...	...	...

图 7B

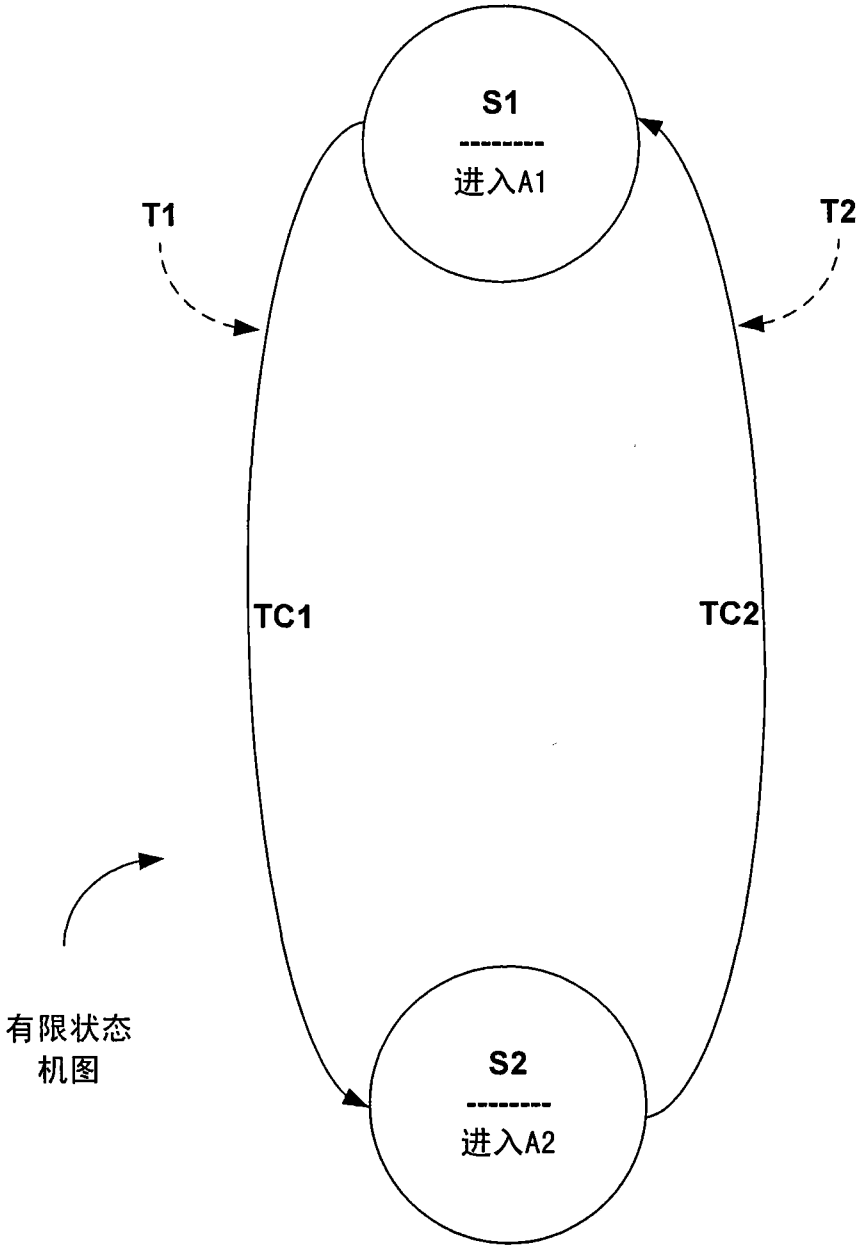


图 8A

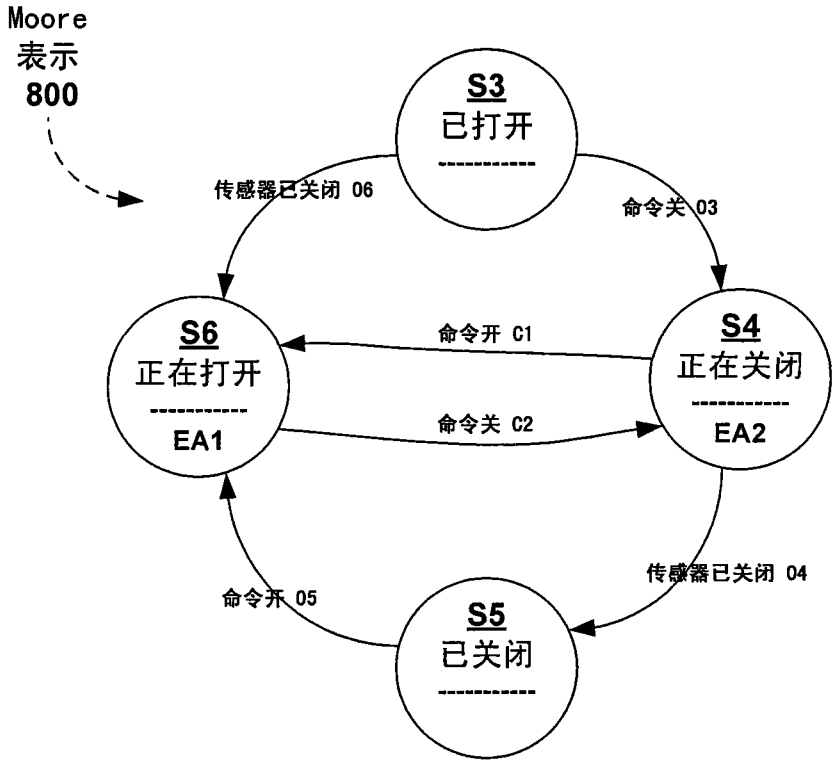

Moore
表示
800
S3
已打开
传感器已关闭 06
命令关 03
S6
正在打开
EA1
命令开 C1
命令关 C2
S4
正在关闭
EA2
传感器已关闭 04
命令开 05
S5
已关闭

图 8B

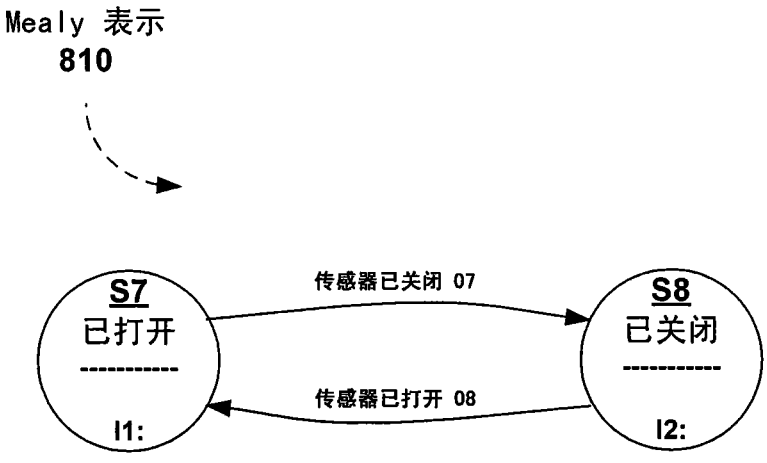

Mealy 表示
810
S7
已打开
I1:
传感器已关闭 07
传感器已打开 08
S8
已关闭
I2:

图 8C

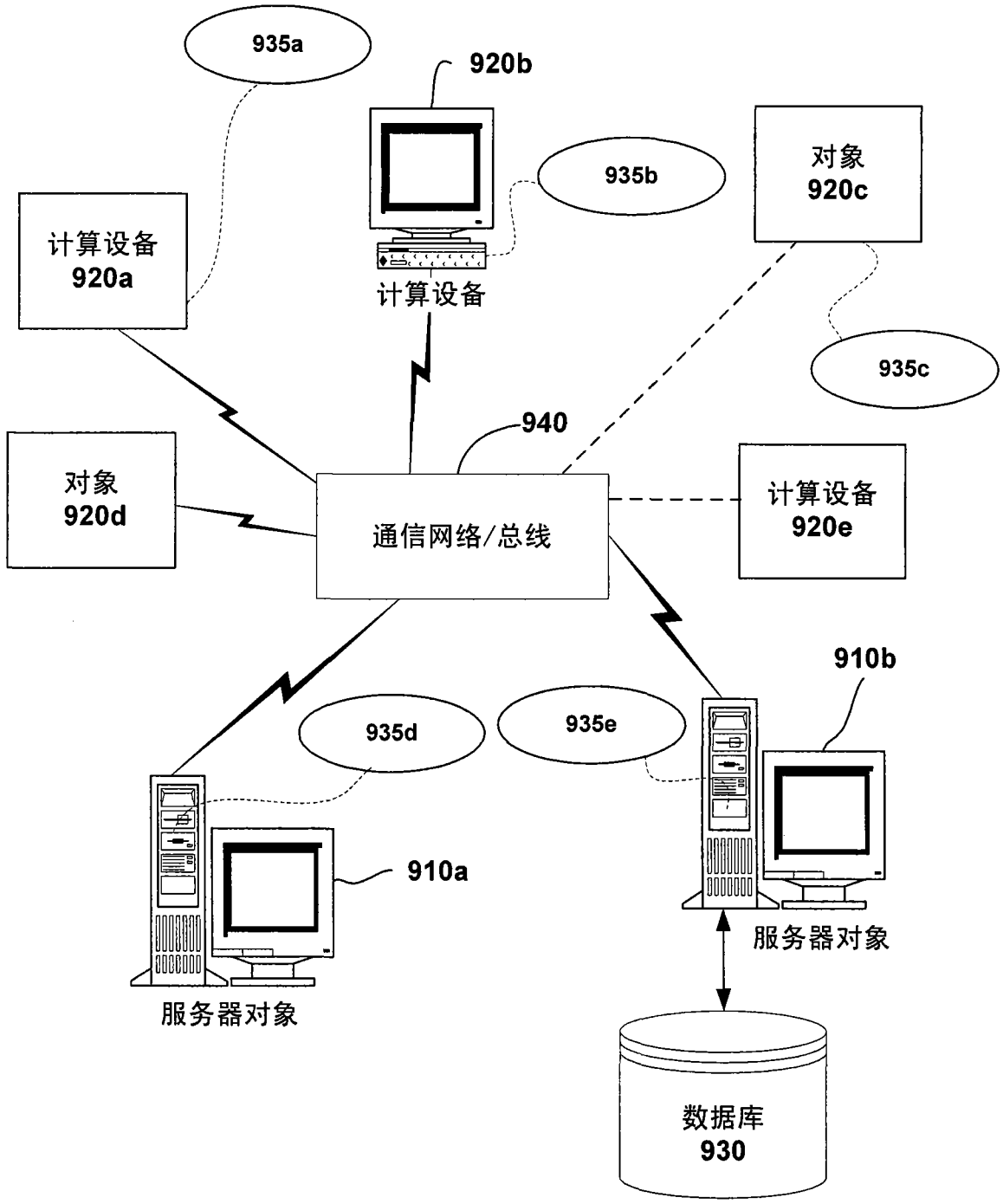

935a
920b
935b
对象
920c
计算设备
920a
计算设备
935c
940
对象
920d
通信网络/总线
计算设备
920e
910b
935d
935e
910a
服务器对象
服务器对象
数据库
930

图 9

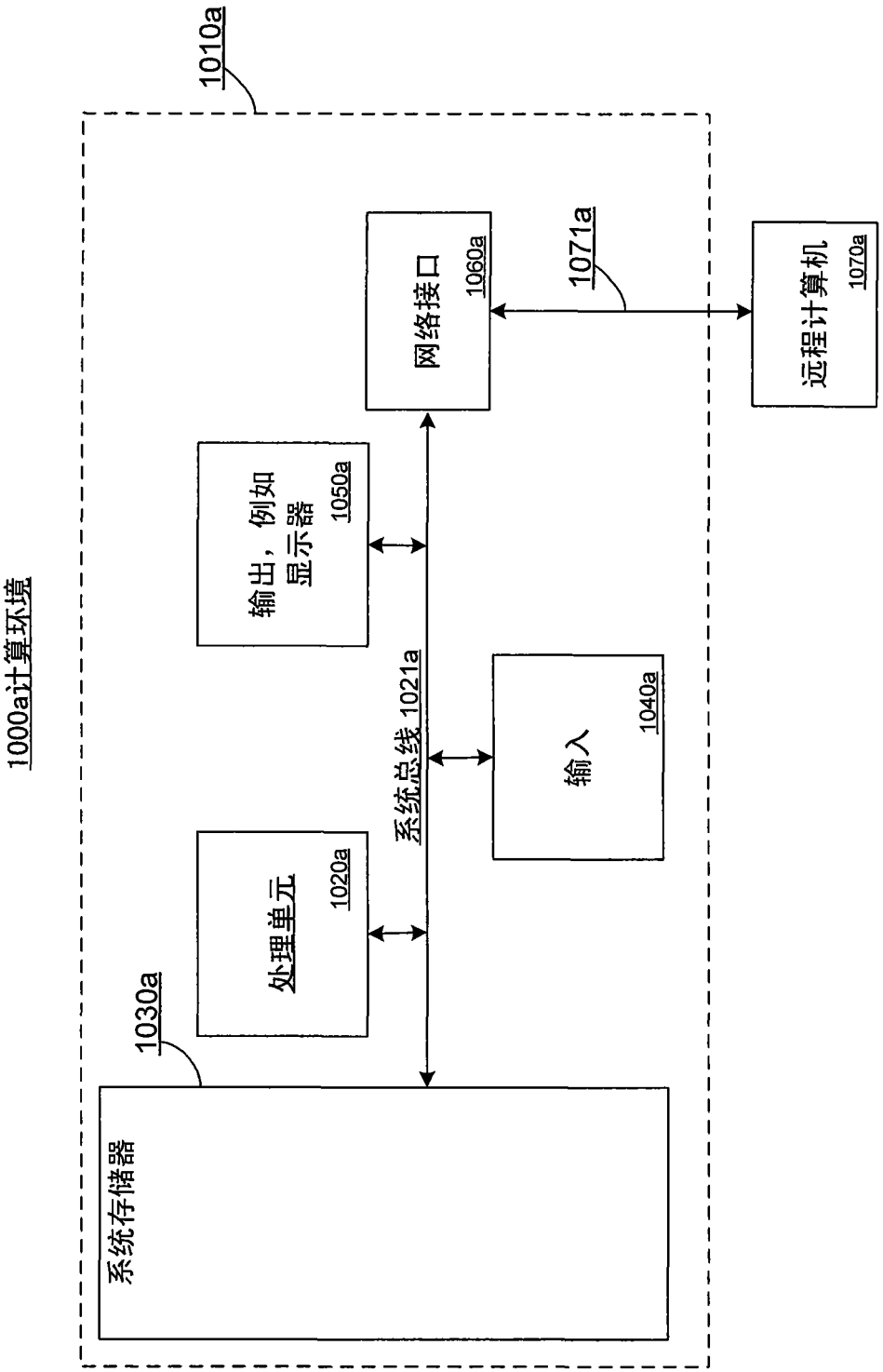

1000a计算环境
1010a
网络接口
1060a
1071a
远程计算机
1070a
输出，例如
显示器
1050a
系统总线 1021a
输入
1040a
处理单元
1020a
1030a
系统存储器

图 10