[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 11/36

[12] 发明专利申请公开说明书

[21] 申请号 03157926.4

[43] 公开日 2004 年 11 月 3 日

[11] 公开号 CN 1542621A

[22] 申请日 2003.8.29 [21] 申请号 03157926.4

[30] 优先权

[32] 2003. 4.28 [33] US [31] 10/424，356

[71] 申请人 英特尔公司

地址 美国加利福尼亚州

[72] 发明人 孙鸣秋

[74] 专利代理机构 北京三友知识产权代理有限公司

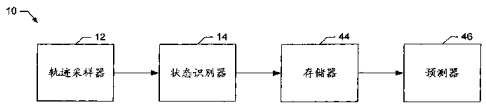
代理人 李 辉

权利要求书 7 页 说明书 16 页 附图 10 页

[54] 发明名称 检测程序模式的方法和设备

[57] 摘要

公布了一种用于检测正在执行的程序中的模式的方法和设备。 所公布的示例性方法构造程序的轨迹；通过比较相互重叠的至少部分地表示与所述轨迹相关的地址的数据集，识别程序状态序列；构造程序状态序列中在各对程序状态之间转换的概率集；识别程序的当前程序状态；和根据当前程序状态和至少一个所述的概率，预测后续的程序状态。



ISSN1008-4274

1．一种检测宏观事务的方法，包括：

构造程序轨迹；

从轨迹中识别程序状态序列；

确定与所述序列中识别的程序状态相关的熵值；和

根据熵值识别宏观事务。

2．如权利要求1所述的方法，其中，所述的构造轨迹包括构造程序计数器轨迹、指令指针轨迹、基本块轨迹和内存地址轨迹中的至少一种。

3．如权利要求1所述的方法，其中，所述的识别程序状态序列包括：

给轨迹中的项目集合分配可能状态签名；

选择一个可能状态签名作为第一状态签名；

将第一状态签名与至少一个后续的可能状态签名进行比较；和

如果所述的至少一个后续的可能状态签名与第一状态签名之间存在至少预定量的差别，则把该后续的可能状态签名识别为第二状态签名。

4．如权利要求3所述的方法，其中，给轨迹中的项目集合分配可能状态签名包括：

由轨迹中的第一项目集合构造第一可能状态签名；和

由轨迹中的第二项目集合构造第二可能状态签名，第一项目集合与第二项目集合部分地重叠。

5．如权利要求4所述的方法，其中，由轨迹中的第一项目集合构造第一可能状态签名包括：

对第一集合的成员进行加权，使后面的成员比前面的成员具有更大的权值；和

将加权后的成员映射为位向量签名。

6．如权利要求3所述的方法，其中，所述的至少一个后续的可能状态签名包括第二状态签名以及第一和第二签名之间插入的至少一个可能签名，并且在识别第一和第二签名后，忽略该至少一个插入的可能签名。

7．如权利要求1所述的方法，其中，确定与序列中识别的程序状态相关的熵值包括：

确定从第一程序状态转换至多个程序状态的概率；和

将所述的概率转换为第一程序状态的熵值。

8．如权利要求7所述的方法，其中，所述的概率包括移动均值和指数移动均值中的至少一种。

9．如权利要求7所述的方法，其中，将概率转换为第一程序状态的熵值包括计算：从第一程序状态转换为第二程序状态的概率（a）和从第一程序状态转换为第二程序状态的概率的对数（b）的乘积（1）和从第一程序状态转换为第三程序状态的概率（a）和从第一程序状态转换为第三程序状态的概率的对数（b）的乘积（2）的和值。

10．如权利要求9所述的方法，其中，将概率转换为第一程序状态的熵值进一步包括将所述和值乘以一个常数。

11．如权利要求1所述的方法，其中，根据熵值识别宏观事务包括：

把熵值中的第一个尖峰识别为宏观事务的结束。

12．如权利要求1所述的方法，进一步包括用各自的首次发现时间来索引各个程序状态。

13．如权利要求1所述的方法，进一步包括预测当前程序状态预期会转换到的至少一个后续的可能程序状态。

14．一种制造物品，其存储有机器可读指令，所述指令执行时使机器：

构造程序的轨迹；

从轨迹中识别程序状态序列；

确定与所述序列中识别的程序状态相关的熵值；和

根据熵值识别宏观事务。

15．如权利要求14所述的制造物品，其中，所述的轨迹包括程序计数器轨迹、指令指针轨迹、基本块轨迹和内存地址轨迹中的至少一种。

16．如权利要求14所述的制造物品，其中，机器可读指令使机器通过以下步骤识别程序状态序列：

给轨迹中的项目集合分配可能状态签名；

选择一个可能状态签名作为第一状态签名；

将第一状态签名与至少一个后续的可能状态签名进行比较；和

如果所述的至少一个后续的可能状态签名与第一状态签名之间存在至少预定量的差别，则把该后续的可能状态签名识别为第二状态签名。

17．如权利要求16所述的制造物品，其中，机器可读指令使机器通过以下步骤给轨迹中的项目集合分配可能状态签名：

由轨迹中的第一项目集合构造第一可能状态签名；和

由轨迹中的第二项目集合构造第二可能状态签名，第一项目集合与第二项目集合部分地重叠。

18．如权利要求17所述的制造物品，其中，机器可读指令使机器通过以下步骤由轨迹中的第一项目集合构造第一可能状态签名：

对第一集合的成员进行加权，使后面的成员比前面的成员具有更大的权值；和

将加权后的成员映射为位向量签名。

19．如权利要求16所述的制造物品，其中，所述的至少一个后续的可能状态签名包括第二状态签名以及第一和第二签名之间插入的至少一个可能签名，并且在识别第一和第二签名后，忽略该至少一个插入的可能签名。

20．如权利要求14所述的制造物品，其中，机器可读指令使机器通过以下步骤确定与序列中识别的程序状态相关的熵值：

确定从第一程序状态转换至多个程序状态的概率；和

将概率转换为第一程序状态的熵值。

21．如权利要求20所述的制造物品，其中，所述的概率包括移动均值和指数移动均值中的至少一种。

22．如权利要求20所述的制造物品，其中，机器可读指令使机器通过计算：从第一程序状态转换为第二程序状态的概率（a）和从第一程序状态转换为第二程序状态的概率的对数（b）的乘积（1）以及从第一程序状态转换为第三程序状态的概率（a）和从第一程序状态转换为第三程

序状态的概率的对数（b）的乘积（2）的和值而将概率转换为第一程序状态的熵值。

23．如权利要求22所述的制造物品，其中，机器可读指令使机器将所述的和值乘以一个常数，从而将概率转换为第一程序状态的熵值。

24．如权利要求14所述的制造物品，其中，机器可读指令使机器把熵值中的第一个尖峰识别为宏观事务的结束，从而根据熵值识别宏观事务。

25．如权利要求14所述的制造物品，其中，机器可读指令使机器根据各自的首次发现时间来索引各个程序状态。

26．如权利要求14所述的制造物品，其中，机器可读指令还使机器预测当前程序状态预期会转换到的至少一个后续的可能程序状态。

27．一种识别程序状态的设备，包括：

轨迹采样器，构造程序的轨迹；

状态识别器，从轨迹中识别程序状态序列；和

预测器，确定与状态识别器识别的程序状态相关的熵值。

28．如权利要求27所述的设备，其中，预测器根据熵值识别宏观事务。

29．如权利要求27所述的设备，其中，所述的轨迹包括程序计数器轨迹、指令指针轨迹、基本块轨迹和内存地址轨迹中的至少一种轨迹。

30．如权利要求27所述的设备，其中，状态识别器进一步包括：

签名构造装置，构造轨迹中的项目集合的可能状态签名；

状态辨别器，根据可能状态签名识别程序状态；和

存储器，存储状态辨别器识别的程序状态的状态签名。

31．如权利要求30所述的设备，其中，签名构造装置通过将轨迹中的第一项目集合映射为第一位向量签名来构造第一可能状态签名。

32．如权利要求31所述的设备，其中，签名构造装置将轨迹第二项目集合映射为第二位向量签名，从而由轨迹中的第二项目集合构造第二可能状态签名

33. 如权利要求 32 所述的设备，其中，第一项目集合与第二项目集合部分地重叠。

34. 如权利要求 31 所述的设备，进一步包括权值分配机构，在将轨迹第一项目集合映射为第一位向量签名时，对第一集合的成员进行加权，使后面的成员比前面的成员具有更大的权值。

35. 如权利要求 30 所述的设备，其中，状态辨别器选择一个可能状态签名作为第一状态签名；将第一状态签名与至少一个后续的可能状态签名进行比较；如果所述至少一个后续的可能状态签名与第一状态签名之间存在至少预定量的差别，则把该后续的可能状态签名识别为第二状态签名，从而识别程序状态。

36. 如权利要求 35 所述的设备，其中，所述的至少一个后续的可能状态签名包括第二状态签名以及第一和第二签名之间插入的至少一个可能签名，并且在识别第一和第二签名后，抛弃该至少一个插入的可能签名。

37. 如权利要求 30 所述的设备，其中，所述的存储器存储数据结构，所述数据结构包括以下的至少一个：第一程序状态的签名；第一程序状态的年龄；第一程序状态的使用频率；第一程序状态的熵值；和从第一程序状态转换至一组程序状态的一组概率。

38. 如权利要求 27 所述的设备，其中，预测器进一步包括：

状态转换监视器，计算从第一程序状态转换至多种程序状态的概率；和

熵值计算器，将所述的概率转换为第一程序状态的熵值。

39. 如权利要求 38 所述的设备，其中，状态转换监视器以指数移动均值计算所述的概率。

40. 如权利要求 39 所述的设备，其中，熵值计算器通过计算：从第一程序状态转换为第二程序状态的概率（a）和从第一程序状态转换为第二程序状态的概率的对数（b）的乘积（1）和从第一程序状态转换为第三程序状态的概率（a）和从第一程序状态转换为第三程序状态的概率的对数（b）的乘积（2）的和值，从而将概率转换为第一程序状态的熵值。

41．如权利要求40所述的设备，其中，熵值计算器将所述的和值乘以一个常数，从而将概率转换为第一程序状态的熵值。

42．如权利要求27所述的设备，其中，预测器进一步包括事件预测器，以根据所述的熵值识别宏观事务。

43．如权利要求42所述的设备，其中，事件预测器把熵值中的第一个尖峰识别为宏观事务的结束。

44．如权利要求27所述的设备，其中，状态识别器和熵值计算器中至少一个根据各自的首次发现时间来索引各个程序状态。

45．如权利要求27所述的设备，其中，预测器进一步包括预测至少一个后续的可能程序状态的事件预测器。

46．一种预测所执行程序中的后续状态的方法，包括：

构造程序的轨迹；

通过比较相互重叠的至少部分地表示与所述轨迹相关的地址的数据集，识别程序状态序列；

构造程序状态序列中在各对程序状态之间转换的概率集；

识别程序的当前程序状态；和

根据当前程序状态和至少一个所述的概率，预测后续的程序状态。

47．如权利要求46所述的方法，其中，识别程序状态序列包括：

给至少部分地表示所述地址的数据集分配可能状态签名；

选择一个可能状态签名作为第一状态签名；

将第一状态签名与至少一个后续的可能状态签名进行比较；和

如果所述至少一个后续的可能状态签名与第一状态签名之间存在至少预定量的差别，则把后续的可能状态签名识别为第二状态签名。

48．如权利要求46所述的方法，其中，所述的地址包括指令地址和内存地址中至少一种。

49．一种预测程序的后续程序状态的设备，包括：

轨迹采样器，构造程序的轨迹；

状态识别器，通过比较相互重叠的至少部分地表示与所述轨迹相关的地址的数据集，识别程序状态序列；

状态转换监视器，计算在各个程序状态之间转换的概率；和

预测器，根据当前状态和至少一个所述的概率预测至少一个后续的可能程序状态。

50．如权利要求49所述的设备，其中，状态识别器进一步包括：

签名构造装置，构造至少部分地表示所述地址的数据集的可能状态签名；

状态辨别器，根据可能状态签名识别程序状态；和

存储器，存储状态辨别器识别的程序状态的状态签名。

51．如权利要求50所述的设备，其中，签名构造装置通过将所述轨迹项目至少部分地表示所述地址的数据集中的第一数据集映射为第一位向量签名，从而构造第一可能状态签名。

52．如权利要求49所述的设备，其中，所述的地址包括指令地址和内存地址中至少一种。

53．一种识别程序状态的系统，包括：

轨迹采样器，构造程序的轨迹；

状态识别器，从轨迹中识别程序状态序列；

预测器，确定与状态识别器识别的程序状态相关的熵值；和

静态随机存取存储器，存储所述的熵值。

54．如权利要求53所述的设备，其中，预测器根据熵值识别宏观事务。

检测程序模式的方法和设备

技术领域

本发明涉及程序管理，具体而言，涉及检测程序中的模式的方法和设备。

背景技术

由计算机和其它基于处理器的设备执行的程序通常会表现出重复的模式（pattern）。众所周知，识别这样的重复模式可以优化程序的执行。例如，软件和固件程序员长期利用小规模的重复模式，如使用迭代循环等，来减小代码规模、控制内存分配以及执行旨在优化和简化程序执行的其它任务。

最近，对寻找识别复杂的工作任务，例如，可控运行时环境（managed run-time environment）和其它基于服务器的应用中的大规模重复模式，以优化这些工作任务的兴趣越来越高。例如，众所周知，一个工作任务可以概念化为一系列的宏观事务。这里使用的术语“宏观事务（macroscopic transaction）”和“子事务（sub-transaction）”是指商业级事务和/或应用软件级事务。例如，互联网零售商，如 Amazon.com 的服务器的工作任务可以概念化为产品展示、订单输入、订单处理、顾客登记、支付处理等一系列宏观事务和子事务。进入更微观一级，工作任务中的各个宏观事务可以看做为一系列的程序状态。希望能够通过，例如，减少主计算机执行宏观事务和/或程序状态和/或在这之间转换所占用的时间来优化工作任务的执行。因此，希望能够识别宏观事务中程序状态的重复模式，以预测程序状态的转换，优化宏观事务和/或程序状态的执行，提高与这些事务相关的工作任务的处理能力。

曾经尝试过利用重复结构（如循环）以把数据预取（prefetch）至高速缓存中。但是，那些现有技术的方法很大程度上局限于高度规则和

简单的工作任务，如科学代码的执行。有效地预测更大、更复杂的工作任务的程序状态和/或宏观事务仍是一个未解决的难题。

附图说明

图 1 是检测程序模式的示例性设备的示意图；

图 2 是图 1 中示例性状态识别器的更详细的示意图；

图 3 是一个示例性轨迹的示意图；

图 4 解释了图 3 所示签名构造装置和权值分配机构构造签名的示例性方式；

图 5 显示了可以为程序中识别的各个状态创建的数据结构的示例；

图 6 是图 1 所示预测器示例的更详细的示意图；

图 7 是由图 6 所示的熵值计算器计算的熵值示例的图表；

图 8 是实现图 1 所示设备的轨迹采样器的机器可读指令示例的流程图；

图 9A-9C 是实现图 1 所示设备的状态识别器和预测器的机器可读指令示例的流程图；

图 10 是可以执行图 8 和图 9A-9C 所示程序以实现图 1 所示设备的计算机示例的示意图。

具体实施方式

如上所述，实际的全球服务器应用程序通常会表现出重复的行为。这些重复行为通常是由本地或远程客户端请求执行主站点的应用程序接口（API）所定义的任务或商业交易而驱动的。由于可供客户端请求执行的任务是有限的，所以在主服务器上客户端的 API 调用是重复性的程序执行模式。如下所述，这种重复性提供了提高效率的机会，可以通过微处理器结构和/或软件而加以利用。

这些重复性程序执行模式中的基本单元是宏观事务或子事务。宏观事务或者子事务可以被认为是一个由指令来度量的路径长度

（pathlength）。例如，这样的事务或者子事务的路径长度通常介于 10^4 至 10^6 条指令之间。

每个事务或子事务包括一个或多个程序状态。程序状态定义为在给定的时间窗口中信息的集合（例如，一组内存地址和/或一组指令地址）。程序状态具有基于测量和可调的特性。另一方面，事务或子事务通常是工作任务的固有特性。

图 1 是示例性设备 10 预测正在执行的程序的程序状态和/或识别程序的宏观事务的示意图。为了构造程序的轨迹（trace），设备 10 具有轨迹采样器 12。轨迹采样器以传统的方式工作，构造程序的任何类型的轨迹。例如，轨迹采样器 12 可以使用硬件计数器（如处理器计数器）和/或软件方法（如可控运行时环境（MRTE）方法从所执行的程序中采集轨迹数据。例如，轨迹采样器 12 可以获取处理器的程序计数器中出现的指令地址，创建指令地址轨迹。作为其它的例子，轨迹采样器 12 可以检测和处理器的高速缓存相关联的地址总线，创建内存地址轨迹。本领域普通技术人员可以容易地理解，可以采用许多其它技术来创建相同或不同类型的轨迹。例如，轨迹采样器 12 也可以配置为创建基本块（basic block）轨迹。

为了由轨迹采样器 12 生成的轨迹而识别程序状态序列，设备 10 还具有状态识别器 14。本领域普通技术人员可以理解，状态识别器 14 可以用任意的方式识别轨迹采样器 12 所生成的（或正在创建的）轨迹内的各个状态。在所示的例子中，状态识别器 14 通过比较相邻的至少部分地表示轨迹中出现的项目（entry）的数据集来识别程序状态。为了使这个比较更可控制，所示的状态识别器 14 将集合转换成位向量，其作为集合中数据的速记代理（shorthand proxy）。然后所示的状态识别器 14 比较相邻集合的位向量并确定位向量之间的差别是否足以说明已经出现了新的状态。每个集合可以包括轨迹中顺序的项目组。可以使用轨迹的所有项目，或使用项目的一个子集（如每十个项目中使用一个），以创建集合。而且，可以使用选进集合中的项目的一部分（如最后八个比特），或该项目的全部（如该项目中的所有比特）来创建位向量。本领域普通

技术人员可以很容易地理解，调节集合的分辨度（如，通过调节在创建集合时跳过的项目数量和/或调节轨迹中用于生成位向量的项目的比特的数目和位置），可以调节由状态识别器 14 识别的程序状态的确定性。因此，程序状态的精确度是基于测量并且可调的。

状态识别器 14 的例子如图 2 所示。在显示的例子中，状态识别器 14 包括签名构造装置（signature developer）16，用于从轨迹的项目集中构造可能状态签名（possible state signature）。为了更好地解释签名构造装置 16 的工作，参见图 3 所示的轨迹示例。在图 3 的例子中，轨迹 18 包括一系列项目，这些项目以某种方式表示计算机和/或其部件的由于程序执行而随着时间改变的特征。例如，这些项目可以是在处理器的程序计数器中出现的指令地址，在与处理器相关联的高速缓存的地址总线上出现的内存地址，或计算机中随程序执行而变化的任何其它可记录特征。本领域普通技术人员可以理解，这些项目可以是完整地址、完整地址的一部分、和/或完整地址或部分地址的代理（proxy）。考虑到被跟踪记录以创建轨迹 18 的项目的数据类型会有很大的范围，图 3 以符号“A”后面跟一个数字来一般性地描述这些项目。符号“A”后面的数字用于唯一地区分这些项目。程序的执行使用来创建轨迹的被监视特征两次或多次具有相同的值，则轨迹 18 会两次或多次包含相同的项目（如，项目 A5 在轨迹 18 中出现两次）。符号“A”后面的数字可以指示该项目相对于其它项目的相对位置。例如，如果轨迹 18 是一个指令地址轨迹，则字母后面的各个数字表示相应地址在内存中的位置。为了便于说明，如果没有特别说明，则下面的例子都假设轨迹 18 是指令地址轨迹，反映运行关心程序的处理器所执行的指令的完整内存地址。

签名构造装置 16 的主要目的是为轨迹 18 中的项目创建代理。具体而言，轨迹 18 中的项目可以包含很多的数据。为了将这些项目转换为更加可管理的表示，签名构造装置 16 将项目归组为集合 26，并将集合 26 转换为可能状态签名 28。在所示的例子中，可能状态签名 28 是位向量。集合 26 可以如图 4 所示转换为位向量 28。

在图 4 的例子中，随机散列函数（random hashing function）30 用于将集合 16 中的项目映射为一个 n 位向量 28。在图 4 的例子中，值“B”32 定义了模型的分辨率（如，集合 26 中被散列函数 30 跳过和/或处理以生成 n 位向量 28 的项目数量）。散列函数 30 将轨迹 18 中的一组项目映射为位向量的基本作用是本领域普通技术人员所熟知（例如，参见 Dhodapkar & Smith, “Managing Multi-Configuration hardware Via Dynamic Working Set Analysis”）。因此，为简短起见，这里不再进一步解释。有兴趣的读者可以参考任何资料，包括上述提到的 Dhodapkar & Smith 的文章，得到进一步的信息。

在将集合 26 的项目映射为位向量签名 28 时，为了对集合 26 的成员进行加权，比如使后面的成员比前面的成员具有更大的权值，设备 10 进一步具有权值分配机构 34。如图 4 的示例性映射函数所示，权值分配机构 34 在使用散列函数 30 对集合 26 进行操作之前，将指数衰减函数（exponential decay function）36 应用于集合 26 的项目（如，$f_1 = e^{-t/T}$，其中 t =时间、T =半寿命周期）。将指数衰减函数应用于集合 26 的项目，因此，当散列函数 30 将集合 26 转换为可能状态签名 28 时，集合 26 中最后面的项目比集合 26 中的早期值对可能状态签名 28 中出现的值具有更大的影响。本领域普通技术人员可以理解，如这里所讨论的其它结构和模块一样，权值分配机构 34 是可选的。换一句话说，图 4 所示的指数衰减函数 36 是可以选择不要的。

如上所述，所示的签名构造装置 16 对轨迹 18 中出现的项目的连续集合 26 进行操作，创建相应于那些集合 26 的一系列位向量 28。本领域普通技术人员可以理解，签名构造装置 16 可以以多种方式将轨迹 18 中的项目归组为集合 26。但是，在所示的例子中，签名构造装置 16 创建集合 26，使相邻的集合发生重叠（即，共享至少一个项目）。换一句话说，签名构造装置 16 使用一个滑动窗口（sliding window）来定义一系列重叠的集合 26。轨迹 18 中由相邻集合 26 共享的项目数量（即，相邻集合的交集）可以小至一个元素，或大至除一个元素之外的所有元素（如，见图 4 中重叠的集合 26）。在签名构造装置 16 创建相邻的相交集合 26

的例子中，使用权值分配机构 34 是特别有利的，这可以使得签名构造装置 16 创建的可能状态签名 28 更对应于新的非重叠项目，而不是重叠项目和旧的非重叠项目。

为了根据可能状态签名 28 识别程序状态，设备 10 进一步具有状态区分器 38。在所示的例子中，状态区分器 38 通过选择一个可能状态签名 28 作为第一状态签名 40（如，图 3 的 STATE1），为其余的分析提供一个参考点，从而开始识别程序状态。通常，默认地选择可能状态签名序列（如，PS1-PSN）中的第一可能状态签名 28（如，图 3 的 PS1）作为第一状态签名 40，但是本领域普通技术人员可以理解，这个选择是任意的，也可以选择另外一个可能状态签名 28（如，PS2-PSN）作为第一状态签名 40。

一旦选择了第一状态签名 40，状态区分器 38 就将第一状态签名 40 与下一个可能状态签名 28（如 PS2）进行比较。例如，如果第一状态签名 40 是第一可能状态签名，则可以把第一状态签名 40 与可能状态签名 28 列表中的第二可能状态签名 PS2 进行比较。如果下一个可能状态签名 28（如 PS2）与第一状态签名 40 之间存在至少预定量的差别，则用于创建轨迹 18 的被测量参数有了足够的变化，说明相应的程序进入了一个新的程序状态。相应地，状态区分器 38 将下一个可能状态签名 28（如 PS2）识别为第二状态签名。

另一方面，如果下一个状态签名 28（如 PS2）与第一状态签名 40 之间没有至少预定量的差别，则用于创建轨迹 18 的被测量参数的变化还不足以说明相应的程序进入了新的程序状态。相应地，状态区分器 38 抛弃可能状态签名 28（如 PS2），跳转至下一个可能状态签名 28（如 PS3），再将第一状态签名 40 与该下一个可能状态签名 28（如 PS3）进行比较，重复上述过程。状态区分器 38 继续这个过程，将可能状态签名 28（如 PS2-PSN）与第一状态签名 40 进行比较，直至识别出一个可能状态签名 28（如 PS4）与第一状态签名 40 之间存在至少预定量的差别。当识别出这样一个可能状态签名（如 PS4）时，状态区分器 38 把这个可能状态签

名（如 PS4）指定为第二状态签名（如 STATE2）。所有参与过的可能状态签名 28（如 PS2-PS3）都不再使用而被抛弃。

一旦识别出第二状态（如 STATE2），状态区分器 38 再开始这个处理，将第二状态签名（如 PS4）和下一个可能状态签名（如 PS5 等）进行比较，以识别第三状态（如 STATE3），直至所有的可能状态签名（如 PS2-PSN）都被检查一遍，从而识别出在程序的当前执行过程中发生的所有程序状态（STATE1 - STATEN）。出现在第一程序状态 40 后面的程序状态例子（如，STATE2 - STATEN）如图 3 所示。如这个例子所示，根据所分析的程序，任何数目的程序状态都可以出现/重复出现任何次数。

本领域普通技术人员可以理解，有许多可能的方法来比较状态签名（如，STATE1 - STATEN）和下一个可能状态签名（如，PS2 - PSN），以确定是否进入了一个新的程序状态。本领域技术人员还可以理解，可以使用许多不同的阈值作为确定是否进入新程序状态的触发器。阈值的选择是在程序中找出的状态的数量和清晰度的决定性因素。在所示的例子中，签名之间表明一个新程序状态所需的阈值之差为汉明距离（Hamming distance）。这样，如果一个状态签名（如，STATE1）和一个可能状态签名（如，PS2）之差满足下式，则已经进入了一个新的程序状态。

Δ = |状态签名 XOR 可能状态签名| / | 状态签名 OR 可能状态签名|

换一句话说，在这个示例性的实施中，如果只出现在（a）当前状态签名和（b）可能状态签名其中一个中的位值集合（即，差值集合）除以出现在(a)当前状态签名和/或(b)可能状态签名中所有成员的集合(即，成员的全集（如，出现在位向量中的逻辑 1 值）的结果大于一个预定值（如，Δ），则进入了一个新的状态。

为了管理与状态区分器 38 识别的状态相关的数据，设备 10 进一步包括存储器 44（见图 1）。所示例子中的存储器 44 是包含多个状态数据结构的状态数组，其中每个数据结构对应于一个唯一的程序状态。本领域普通技术人员可以理解，状态数据结构和状态数组 44 可以以多种形式

设置。在所示的例子中，状态数组 44 很大，足以包含四百个状态数据结构，并且状态数组中的各个数据结构又包括如下的字段：（a）相应程序状态的状态签名，（b）相应程序状态的年龄，（c）相应程序状态的使用频率，（d）相应程序状态的熵值，和（e）包含从相应程序状态转换为一系列程序状态的一系列概率的子数组。

状态数据结构的例子如图 5 所示。状态签名字段可以用来存储相应于数据结构的状态的位向量签名（如，STATE1 - STATEN）。年龄字段可以用来存储指示最后一次进入相应状态的时间的值。因为状态数组是有限的，年龄字段可以用来识别旧的状态数据结构，可以重写旧状态数据结构而存储新发生的状态数据结构。使用频率字段可以用来存储用于识别在该数据结构的寿命周期中进入相应状态的次数的数据。熵值字段可以用来存储用于识别宏观事务结束的数据。概率集子数组可以用来存储指示在该状态数据结构的寿命周期中，程序从对应于该状态数据结构的程序状态进入到各个程序状态的次数比率的数据。例如，每个数据结构可以存储多达十六组的三个字段，包括指示相应于该状态数据结构的程序状态在过去已经转换至的程序状态的名称的数据，那些转换发生的相对时间，和相应于该状态数据结构的程序状态转换至该字段集合中第一个字段中确定的状态的次数百分比。

为了确定和状态识别器所识别的程序状态相关的熵值，设备 10 进一步包括预测器 46。如下所述，在所示的例子中，预测器 46 使用熵值来识别宏观事务的结束。

图 6 更加详细地显示了一个示例性的预测器 46。为了计算从一个程序状态转换至另一个程序状态的概率，预测器 46 具有状态转换监视器 48。每当发生了程序状态转换（也就是，每当程序状态从一种状态变化至另一种状态），状态转换监视器 48 在相应于正在退出的程序状态的状态数据结构的子数组中记录该事件。具体而言，状态转换监视器 48 记录下指示所转换至的数组的名称和转换的发生时间（或时间的代理）的数据。要记录转换发生的时间（或时间的代理），因为在所示的例子中，状态转换监视器 48 使用指数移动平均（exponential moving average）计算

概率。这样，不是简单地对状态数据结构的子数组中的项目进行平均来根据过去的表现计算特定状态之间转换的概率，状态转换监视器 48 根据它们的相对发生时间，通过将那些项目乘以一个指数函数来对状态数据结构的子数组中的项目进行加权。这种方法的结果是，在概率计算中，在时间上后发生的子数组的项目比在时间上早发生的项目有更大的权值，状态转换监视器 48 可以比使用直接移动平均方法更快地识别概率的变化模式。

为了将状态转换监视器 48 计算的概率转换为熵值，设备 10 进一步包括熵值计算器 50。给定状态的熵值是与那个状态相关的转换不确定性。换一句话说，给定当前状态的历史信息，熵值将当前程序状态结束时要发生哪种程序状态的信息不确定性量化。例如，对于给定的具有转换为第二程序状态和第三程序状态的历史信息的程序状态，熵值计算器 50 计算：从本程序状态转换为第二程序状态的概率（a）和从本程序状态转换为第二程序状态的概率对数（b）的乘积（1）和从本程序状态转换为第三程序状态的概率（a）和从本程序状态转换为第三程序状态的概率对数（b）的乘积（2）之和，从而对于给定的程序状态将概率转换为熵值。用另一种方式描述，对于状态数组 44 中的每个状态数据结构，熵转换器 50 根据公知的香农（Shannon）公式计算熵值：

$$H = -K \sum (Pi * \log Pi),$$

其中 H 为熵值，K 为常数，Pi 是从当前状态（也就是，与状态数据结构相关的状态）转换为状态“i”（也就是，在当前状态的数据结构的子数组中识别出的状态）的概率。在所执行的程序中识别出的每个状态的熵值存储在相应状态的数据结构中（见图 5）。

为了预测要从当前状态转换至的下一个可能的程序状态，预测器 46 进一步包括事件预测器 54。事件预测器 54 比较当前程序状态的数据结构的子数组中出现的概率，确定后面最可能的一个或多个状态。后面的最可能的状态是具有最高概率的状态。

事件预测器 54 还根据与当前程序状态相关的熵值来识别宏观事务。从宏观应用的逻辑层次来看，可以观察到与所计算的熵值（H）的连接，

即微观轨迹特征。当新的商业交易开始时，程序执行通常会沿着具有低熵的相对明确的轨线进行。但是，当程序执行到达宏观事务中最后的程序状态时，熵值出现峰值，关于程序将会转换至下一个可能的程序状态有最大的不确定性。换一句话说，在宏观事务内，通常会有重复的程序状态序列。通过观察程序状态之间过去的行为，可以检测到这些模式，并使用它们来预测未来的行为。相反，宏观事务的顺序比宏观事务内程序状态的顺序具有更高程度的随机性，因为执行宏观事务的顺序取决于从第三方接收到的事务请求的顺序，因此具有很大的随机性。为了使这点更清楚，以在线零售商为例。在线零售商的服务器从大量的不同客户端接收请求，并以总体随机的方式将这些请求排列为一个队列。因此处理这些请求的顺序是随机的。但是，一旦服务器开始处理一个请求，它通常会在处理队列中其它事务之前处理完这整个事务。因此，宏观事务末尾处的程序状态通常具有高熵值（也就是，对于要进入的程序状态具有很高的不确定性），因为对于在刚执行完的当前事务之后要执行哪个宏观事务具有很高的不确定性。从而，宏观事务的最后程序状态相对于周围的熵值具有熵值尖峰。换一句话说，宏观事务的最后程序状态的熵值与紧接着该最后程序状态之前或之后的程序状态的熵值相比通常具有相对最大值。

事件预测器 54 利用这个特征，使用熵值尖峰作为宏观事务结束的划分标识。这样，宏观事务可以定义为具有熵值尖峰结束状态的程序状态有序序列。宏观事务映射为商业或应用软件事务，是工作任务的固有特征。相同的宏观事务可以包含不同的程序状态集，这是工作任务的基于测量的特征，可以通过转换阈值进行调节。但是，要注意，对于高级别的商业逻辑并不重要的可重复子事务也会在呈现熵值尖峰的程序状态中结束，这样就可能会错误地识别为宏观事务。这种错误识别在实际情况，如程序的调节中并不是问题，因为对于所有的实际应用，具有很高的转换不确定性的子事务都和事务一样。

如上所述，事件预测器 54 把一系列程序状态的熵值尖峰识别为宏观事务的结束。本领域普通技术人员可以理解，事件预测器 54 可以使用多

种技术来识别熵值尖峰。例如，事件预测器 54 可以将当前状态的熵值与前一个状态的熵值和后一个状态的熵值进行比较。如果当前状态的熵值大于前一个状态的熵值和后一个状态的熵值，则当前状态的熵值就是相对最大值（也就是，尖峰）并且把当前状态识别为宏观事务的结束。否则，当前状态的熵值就不是相对最大值，当前状态不识别为宏观事务的结束。

显示由熵值计算器 50 计算的熵值图表如图 7 所示。在图 7 的图表中，不使用签名来标示程序状态，我们使用每个程序状态的第一发现时间作为唯一的索引。这些第一发现时间作为图 7 的 Y 轴坐标，（Y 轴坐标也表示下面描述的熵值），内存访问作为图 7 的 X 轴坐标。内存访问是时间的代理。

图 7 的图表包括两个图表。一个图表表示在一段时期中进入的程序状态。另一个图表表示同一时期中相应程序状态的熵值。从图 7 中可以看出，图表中的每个状态（也就是，由◆表示的各个数据点）与其相应的熵值（也就是，由正方形■表示的各个数据点）垂直对齐。从图 7 中还可以看出，熵值尖峰是周期性的。每个熵值尖峰代表一个宏观事务的结束。

实现图 1 所示设备 10 的示例性机器可读指令的流程图如图 8 和图 9A-9C 所示。在这个例子中，机器可读指令包括由处理器（如以下结合图 10 所述的示例性计算机 1000 中的处理器 1012）执行的程序。该程序可以包含在存储于有形的介质，如 CD-ROM、软盘、硬盘、数字通用光盘(DVD)、或与处理器 1012 相联的存储器中的软件中，但本领域普通技术人员可以理解，该程序的全部和/或部分也可以以公知的方式由处理器 1012 之外的设备执行，和/或包含在固件或专用硬件中。例如，可以由软件、硬件和/或固件来实现轨迹采样器 12、状态识别器 14、预测器 46、权值分配机构 34、签名构造装置 16、状态区分器 38、状态转换监视器 48、熵值计算器 50、和/或事件预测器 54 中的任何一个或全部。而且，尽管该示例性程序是参照图 8 和图 9A-9C 的流程图描述的，但本领域普通技术人

员可以理解，也可以使用其它的实现设备 10 的方法。例如，各模块的执行顺序可以变化，和/或上述的一些模块可以改变、删掉、或组合。

图 8 的程序从块 100 开始，目标程序开始执行。当目标程序执行时，轨迹采样器 12 创建所执行程序的)一个或多个特征的一个或多个轨迹(块 102）。例如，轨迹采样器 12 可以创建指令地址轨迹、内存地址轨迹、基本块轨迹、和/或任何其它类型的轨迹。控制从块 102 到块 104。

如果已经调用了轨迹处理线程（thread）（块 104），则控制从块 104 到块 106。如果完成了程序的轨迹 18（块 106），则图 8 的程序终止。否则，如果未完成轨迹 18（块 106），则控制返回至块 102，继续轨迹 18 的记录。

如果还没有调用轨迹处理线程（块 104），则控制进行到块 108。在块 108，开始轨迹处理线程。然后控制返回至块 106。如上所述，如果完成了轨迹 18（块 106），则程序终止，或如果未完成轨迹 18 则程序继续生成轨迹 18（块 102）。这样控制继续在块 100-108 之间循环，直至目标程序停止执行并且完成了轨迹 18。

轨迹处理线程的例子如图 9A-9C 所示。所示的轨迹处理线程从块 120 开始，此处签名构造装置 16 从轨迹采样器 12 创建的轨迹 18 中获得项目的集合 26。如上所述，可以以任何方式创建项目的集合 26，包含任何成员。在图 3 的例子中，每个集合 26 包括一系列连续项目（也就是，没有项目被跳过），相邻的集合相互重叠（也就是，至少一个项目被用在两个相邻的集合中）。但是，也可以采用跳过轨迹 18 中的一些项目的集合和/或不重叠的集合。

一旦从轨迹 18 中提取出创建集合 26 的项目（块 120），权值分配机构 39 就调节提取出的项目的值，使后期的项目比早期项目具有更大的权值（块 122）。例如，权值分配机构 34 可以对集合的项目应用指数衰减函数 36（如，$f1 = e^{-t/T}$）（块 122）。

权值分配机构 34 对项目值进行了加权之后（块 122），签名构造装置 16 将集合 26 中的项目映射为一个 n 位向量，以创建集合 26 的可能状

态签名 28（块 124）。如上所述，可以使用散列函数来把集合 26 中的项目映射为可能状态签名 28。

生成可能状态签名 28 后（块 124），状态区分器 38 确定可能状态签名 28 是否是第一可能状态签名（块 126）。如果是第一可能状态签名（块 126），则默认地，把第一可能状态签名定义为第一状态签名。这样，状态区分器 38 把当前状态签名变量设置与等于可能状态签名 28（块 128），并且在第一状态的状态数组 44 中创建状态数据结构（块 130）。状态数据结构的一个例子如图 5 所示。状态区分器 38 通过创建如图 5 所示的字段，将当前状态签名写入新状态数据结构的状态签名字段中，把新状态数据结构的年龄字段设置为等于当前时间或当前时间的代理，并把熵值字段和概率子数组字段设置为等于零，从而创建状态数据结构。

然后，签名构造装置 16 采集下一个项目集合 26 以创建可能状态签名 28（块 132）。在所示的例子中，被签名构造装置 16 用来创建可能状态签名 28 的集合 26 是重叠的。这样签名构造装置 16 可以通过从最近的项目集合 26 中去掉最老的项目并且加入同样数量的新项目来创建新的当前集合 26，以创建下一个集合 26（块 132）。然后控制返回至块 122，如上所述对新的当前集合中的项目进行加权。

在块 126，如果当前的可能状态签名不是第一个可能状态签名，则控制从块 126 跳至块 134（图 9B）。在块 134，状态区分器 38 计算当前状态签名（即，上述的当前状态签名变量的值）和当前可能状态签名之间的差值。状态区分器 38 然后将计算的差值与一个阈值（如，哈明距离）进行比较。如果计算的差值超过该阈值（块 136），则程序状态发生了变化，并且控制进行至块 138。如果计算的差值未超过该阈值（块 136），则如上所述，签名构造装置 16 采集下一个项目集合 26 以创建可能状态签名 28（块 132，图 9A），并且控制返回至块 122。这样，控制继续在块 122-136 之中循环，直至程序状态发生变化。

为便于说明，假设程序状态发生了变化（块 136），状态区分器 38 把当前状态签名变量设置为等于当前可能状态签名 28（块 138）。状态区分器 38 然后检查状态数组 44 中的签名以确定当前状态签名是否对应

于一个已知状态的签名（块 140）。如果当前状态签名是一个已知状态签名，则控制进行至块 160（图 9C）。否则，如果当前状态签名不是已知状态签名（即，当前状态签名不对应于状态数组 44 中已经存在的状态），则控制进行至块 142（图 9B）。

为便于说明，假设当前状态签名是一个未知的状态签名（如，当前程序状态是一个新的程序状态）（块 140），如以上结合块 130 所述，状态区分器 38 在第一状态的状态数组 44 中创建状态数据结构（块 142）。

状态转换监视器 48 然后更新最近状态的概率子数组以反映从最近状态至新的当前状态的转换（块 144）。然后控制进行至块 146，状态区分器 38 确定状态数组 44 是否已满（即，新添加的数据结构是否使用了状态数组中最后的可用位置）。如果状态数组 44 未满，则控制返回至块 132（图 9A），签名构造装置 16 采集下一个项目集合 26 以创建可能状态签名 28。然后如上所述，控制返回至块 122。

如果状态数组已满（块 146），则控制进行至块 150（图 9B），状态区分器 38 从状态数组 44 中删除最旧的状态数据结构。通过比较状态数组 44 中的状态数据结构的使用字段可以识别最旧的状态数据结构。删除最旧的状态数据结构之后（块 150），控制进行至块 148，签名构造装置 16 采集下一个项目集合 26 以创建可能状态签名 28。然后如上所述，控制返回至块 122。

假设当前状态签名是已知的状态签名（块 140），控制进行至块 160（图 9C）。状态转换监视器 48 更新最近状态的概率子数组以反映从最近状态至新的当前状态的转换（块 160）。控制然后进行至块 162，熵值计算器 50 计算当前状态的熵值。如上所述，可以通过许多不同的方式计算熵值。例如，在所示的例子中，使用香农公式计算熵值。

一旦计算了熵值（块 162），事件预测器 54 通过，例如，比较当前状态的状态数据结构的概率子数组中的各个值来识别接下来最可能的状态（块 164）。事件预测器 54 可以检查最近几个状态的熵值来确定是否出现了熵值尖峰（块 168）。如果识别出熵值尖峰（块 168），则事件预测器 54 把对应于熵值尖峰的程序状态识别为宏观事务的最近状态（块

170）。如果未识别出熵值尖峰（块 168），则未发生宏观事务的结束。相应地，控制跳过块 170 而返回至块 132（图 9A）。

不管控制是经过块 170 还是直接由块 168 到达块 132，在块 132，签名构造装置 16 采集下一个项目集合 26 以创建可能状态签名 28。然后如上所述，控制返回至块 122。控制继续在块 122-170 之间循环，直至处理完全部的轨迹 18。一旦处理完全部的轨迹 18，图 9A-9C 的轨迹处理线程结束。

图 10 是能够实现这里所述设备和方法的示例性计算机 1000 的框图。计算机 1000 可以是，例如，服务器、个人计算机、个人数字助理（PDA）、互联网设备、DVD 播放器、CD 播放器、数字摄像机、个人视频记录器、机顶盒、或任何其它类型的计算设备。

这个示例的系统 1000 包括处理器 1012。例如，处理器 1012 可以是 Pentium®系列、Itanium®系列、XScale®系列或 Centrino™系列中的一种或多种 Intel®微处理器。当然，其它系列的其它处理器也适用。

处理器 1012 通过总线 1018 与主存储器（包括易失性存储器 1014 和非易失性存储器 1016）进行通讯。易失性存储器 1014 可以是同步动态随机存取存储器（SDRAM），动态随机寻址存储器（DRAM），RAMBUS 动态随机存取存储器（RDRAM）和/或任何其它类型的随机存取存储设备。非易失性存储器 1016 可以是闪存（flash memory）和/或任何其它适当类型的存储设备。通常由存储控制器（未显示）以传统的方式控制对主存储器 1014、1016 的访问。

计算机 1000 还包括传统的接口电路 1020。接口电路 1020 可以是任何公知类型的接口标准，如以太网接口、通用串行总线（USB）、和/或第三代输入/输出（3GIO）接口。

一个或多个输入设备 1022 连接到接口电路 1020。输入设备 1022 允许用户向处理器 1012 输入数据和命令。输入设备可以是，例如，键盘、鼠标、触摸屏、轨迹板、轨迹球、等位点（isopoint）和/或语音识别系统。

一个或多个输出设备 1024 也连接到接口电路 1020。输出设备 1024 可以是，例如，显示设备（液晶显示器、阴极射线管显示器（CRT）、打印机和/或扬声器）。接口电路 1020 通常包括图形驱动卡。

接口电路 1020 也包括通讯设备，如调制解调器或网络接口卡，以便于经过网络 1026（如，以太网连接、数字用户线路（DSL）、电话线、同轴电缆、蜂窝电话系统等）与外部计算机进行数据交换。

计算机 1000 也包括一个或多个大容量存储设备 1028，用于存储软件和数据。大容量存储设备 1028 包括软盘驱动器、硬盘驱动器、CD 驱动器和数字通用光盘（DVD）驱动器。大容量存储设备 1028 可以实现存储器 44 的功能。

作为在图 10 所示设备这样的系统中实现这里所述的方法和/或设备的另一种选择，这里所描述的方法和/或设备也可以包含在处理器和/或 ASIC（专用集成电路）这样的结构中。

如上所述，本领域普通技术人员可以理解，上述方法和设备可以在静态编译器、可控运行时环境即时编辑器（JIT）中实现、和/或直接在微处理器的硬件中实现，以在执行不同程序时实现性能优化。

尽管以上描述了特定的示例性方法、设备和制造物品，但本发明的不限于此。相反，本发明涵盖落入所附权利要求及其等同物所限定的范围内的所有方法/设备和制造物品。

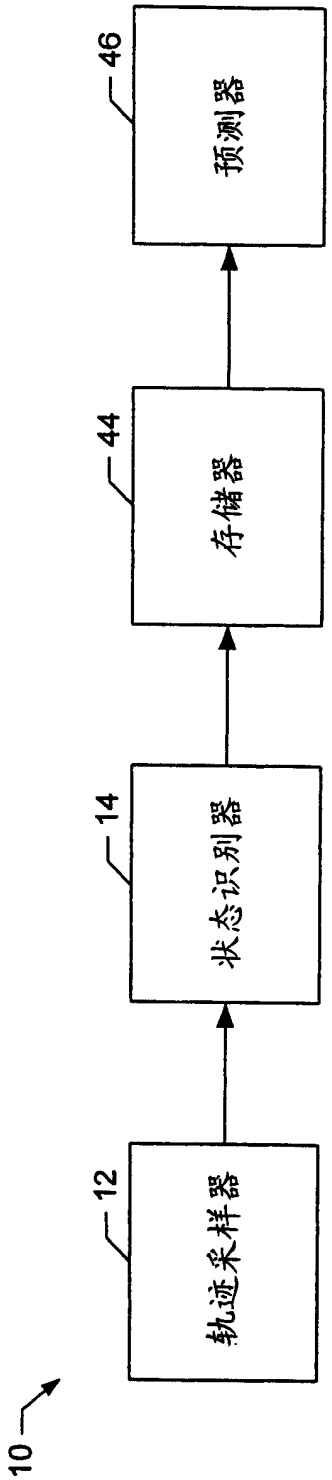

10
12
轨迹采样器
14
状态识别器
44
存储器
46
预测器

图 1

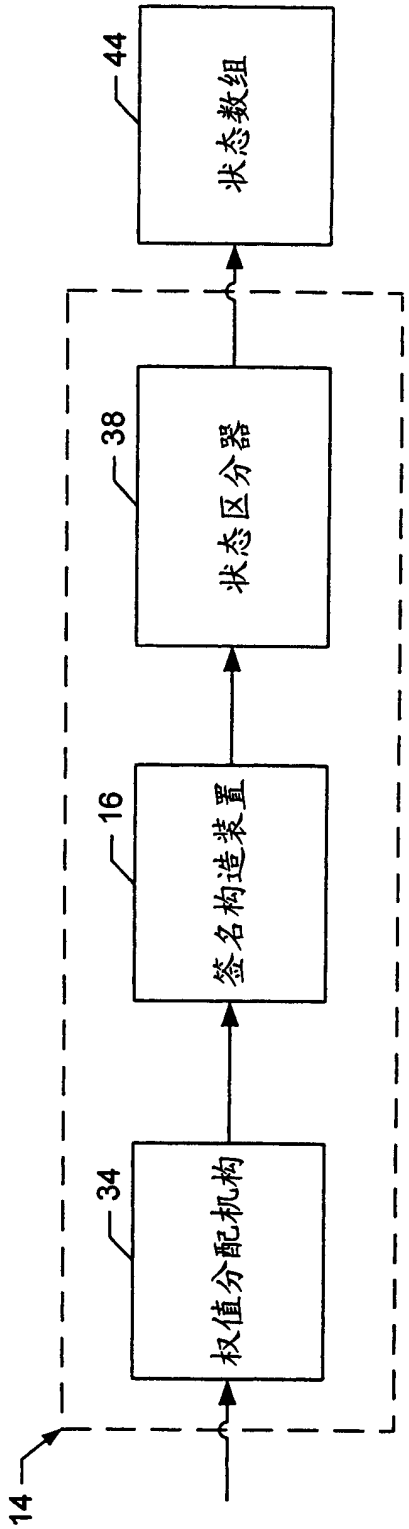

14
34
权值分配机构
16
签名构造装置
38
状态区分器
44
状态数组

图 2

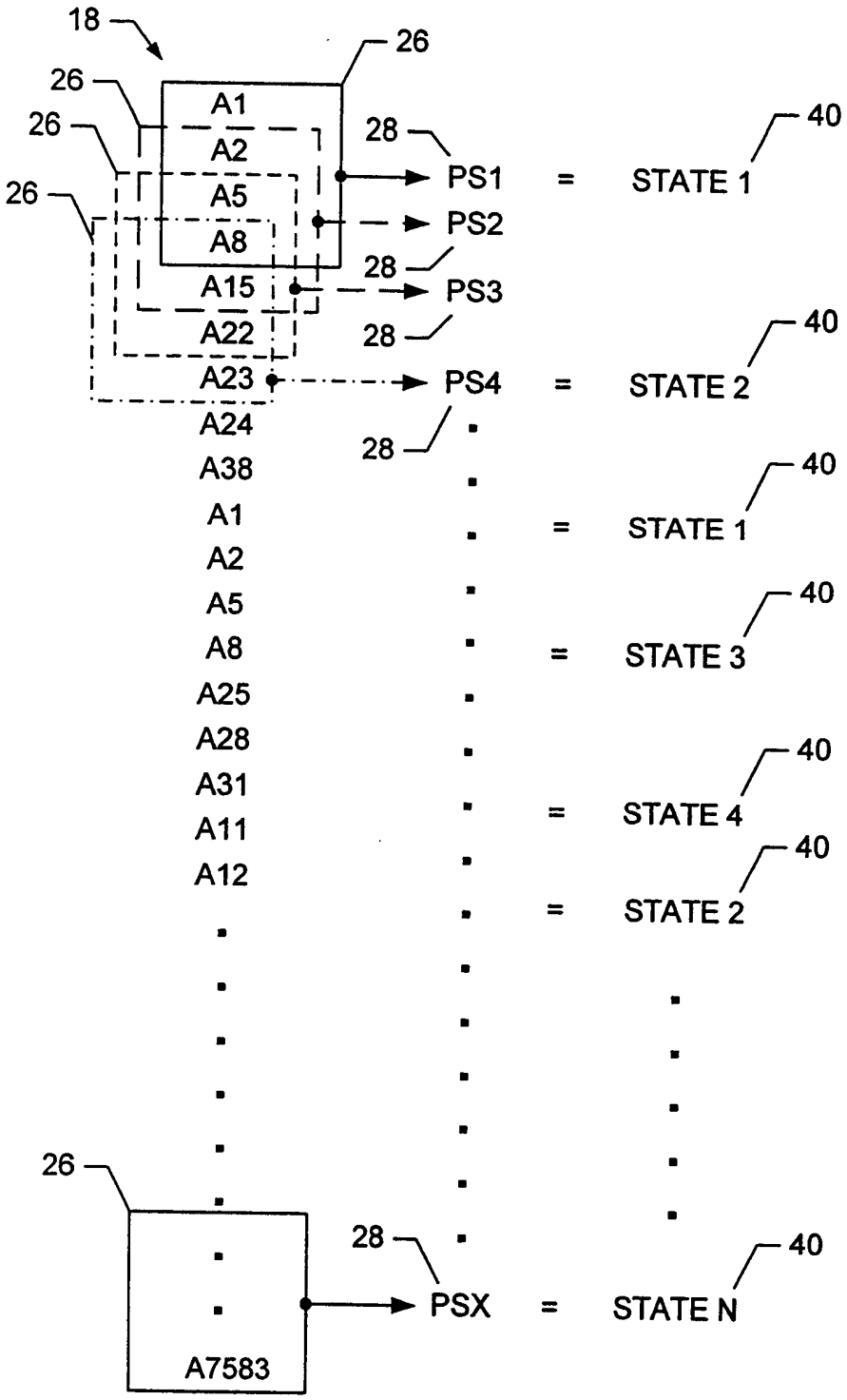

18
26
26
26
26
26
A1
A2
A5
A8
A15
A22
A23
A24
A38
A1
A2
A5
A8
A25
A28
A31
A11
A12
A7583
28
28
28
28
28
PS1
PS2
PS3
PS4
PSX
=
=
=
=
=
=
STATE 1
STATE 2
STATE 1
STATE 3
STATE 4
STATE 2
STATE N
40
40
40
40
40
40

图 3

状态签名:			0101100101110		
年龄:			X		
使用频率:			Y		
熵值:			Z		
状态:	A	出现时间:	00:00:00; 00:05:13	概率:	0.63
状态:	C	出现时间:	00:02:04; 00:09:42	概率:	0.12
状态:	E	出现时间:	00:00:13; 00:02:43	概率:	0.08
状态:	G	出现时间:	00:08:06; 00:09:18	概率:	0.05
状态:	L	出现时间:	00:10:03	概率:	0.02
状态:	M	出现时间:	00:07:53	概率:	0.02
. . .	. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .
. . .	. . .	. . .	. . .	. . .	. . .
状态:	V	出现时间:	00:13:31; 00:15:37	概率:	0.01

图 5

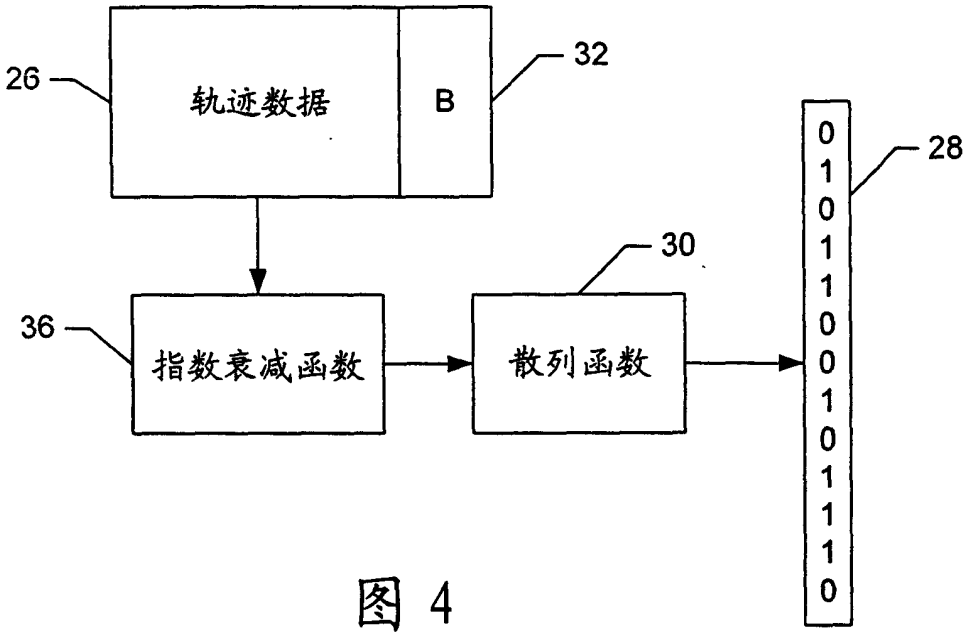

26
轨迹数据
B
32
28
0
1
0
1
1
0
0
1
0
1
1
1
0
30
36
指数衰减函数
散列函数

图 4

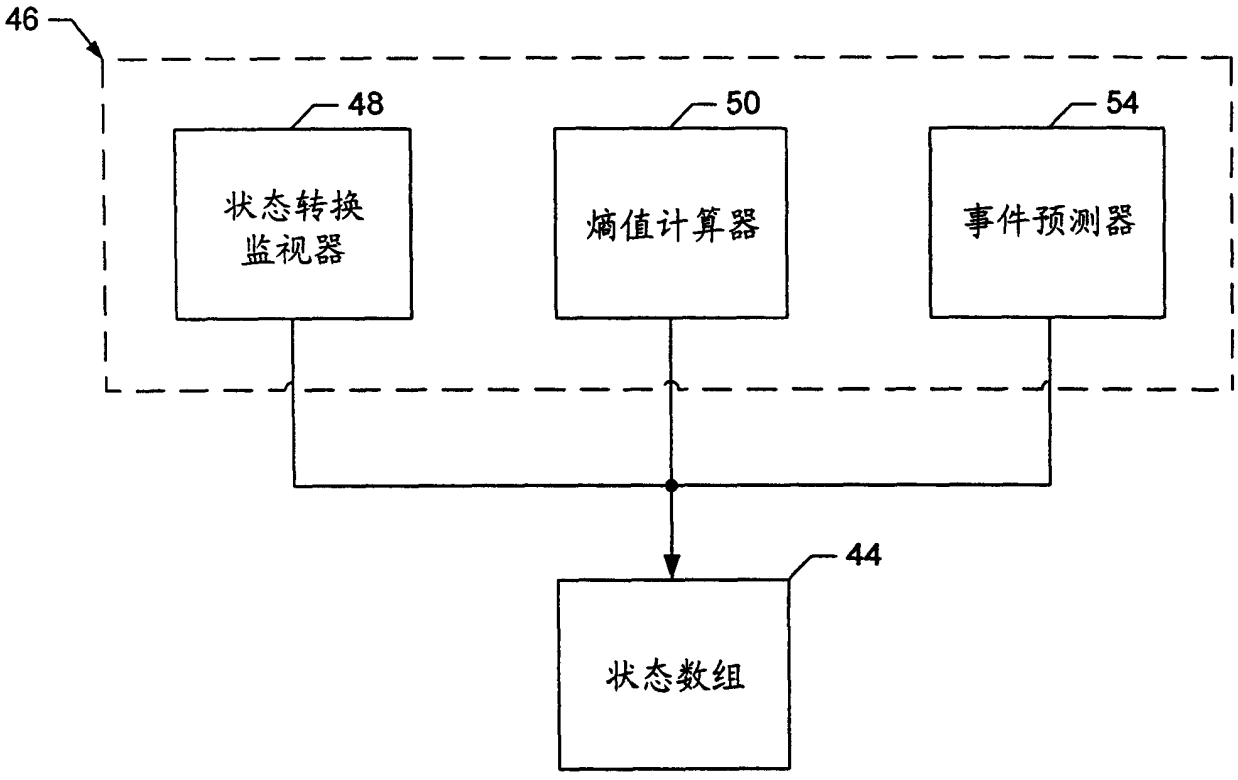

46
48
状态转换
监视器
50
熵值计算器
54
事件预测器
44
状态数组

图 6

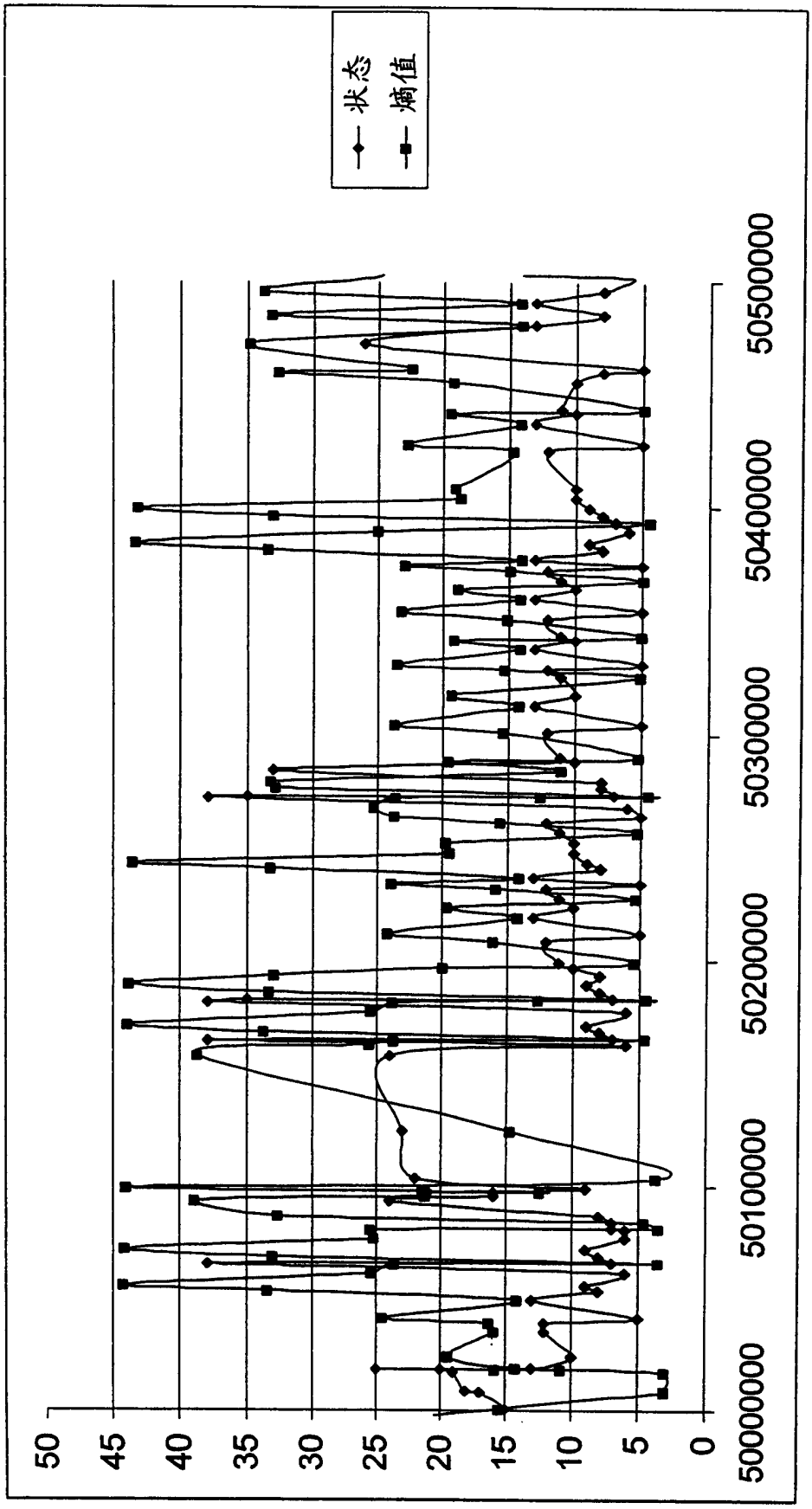

状态
熵值
50
45
40
35
30
25
20
15
10
5
0
5000000
50100000
50200000
50300000
50400000
50500000

图 7

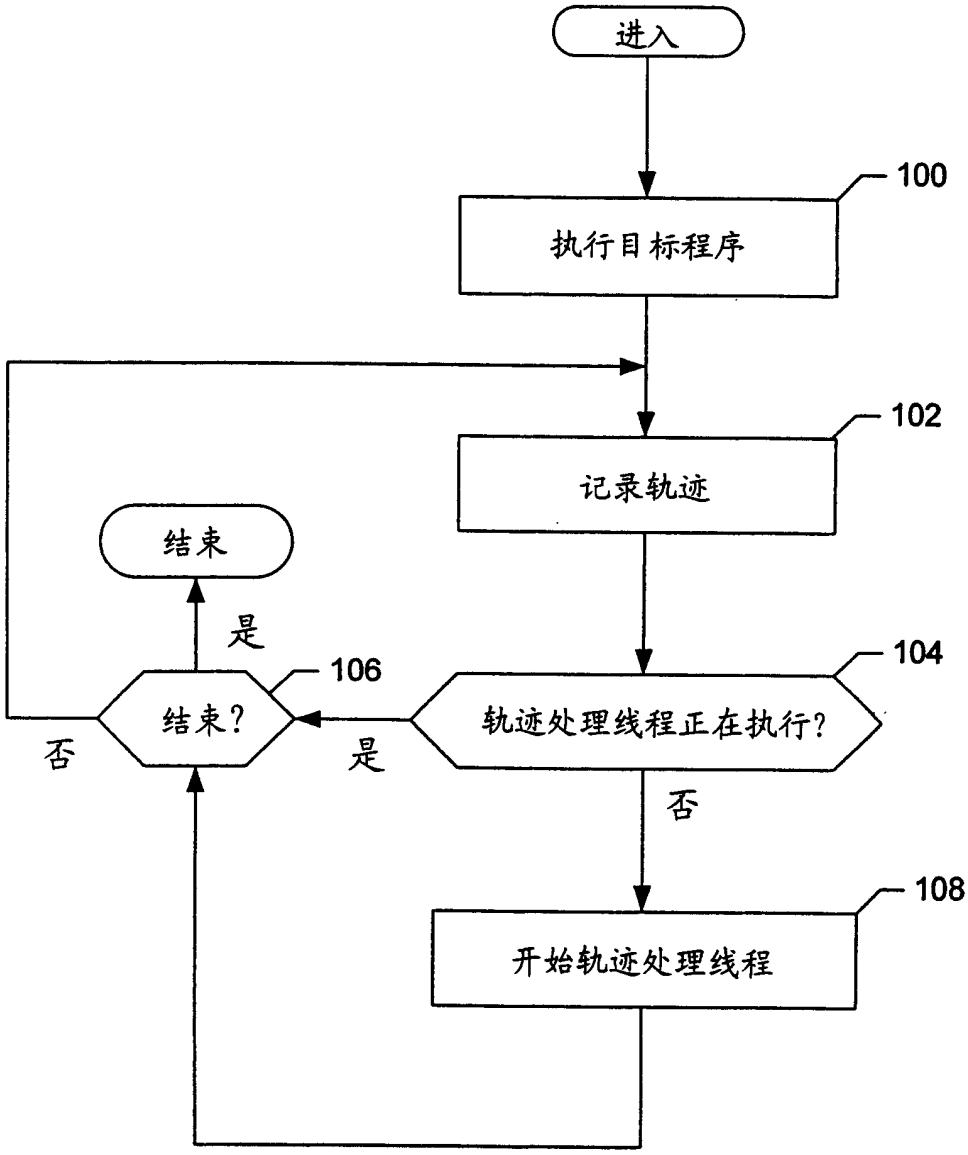

进入
100
执行目标程序
102
记录轨迹
结束
是
106
结束?
否
是
104
轨迹处理线程正在执行?
否
108
开始轨迹处理线程

图 8

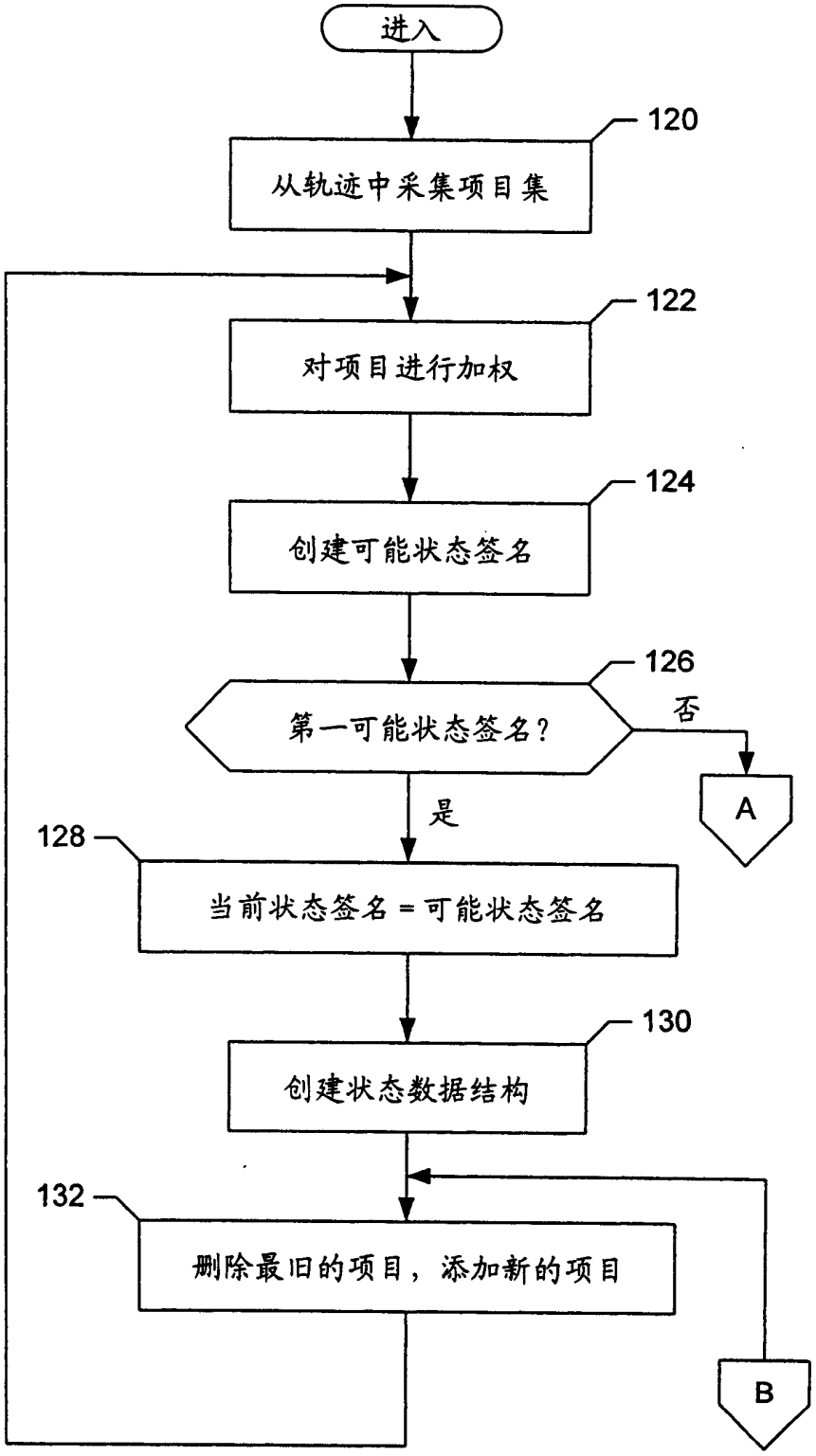

进入
120
从轨迹中采集项目集
122
对项目进行加权
124
创建可能状态签名
126
第一可能状态签名？
否
A
是
128
当前状态签名 = 可能状态签名
130
创建状态数据结构
132
删除最旧的项目，添加新的项目
B

图 9A

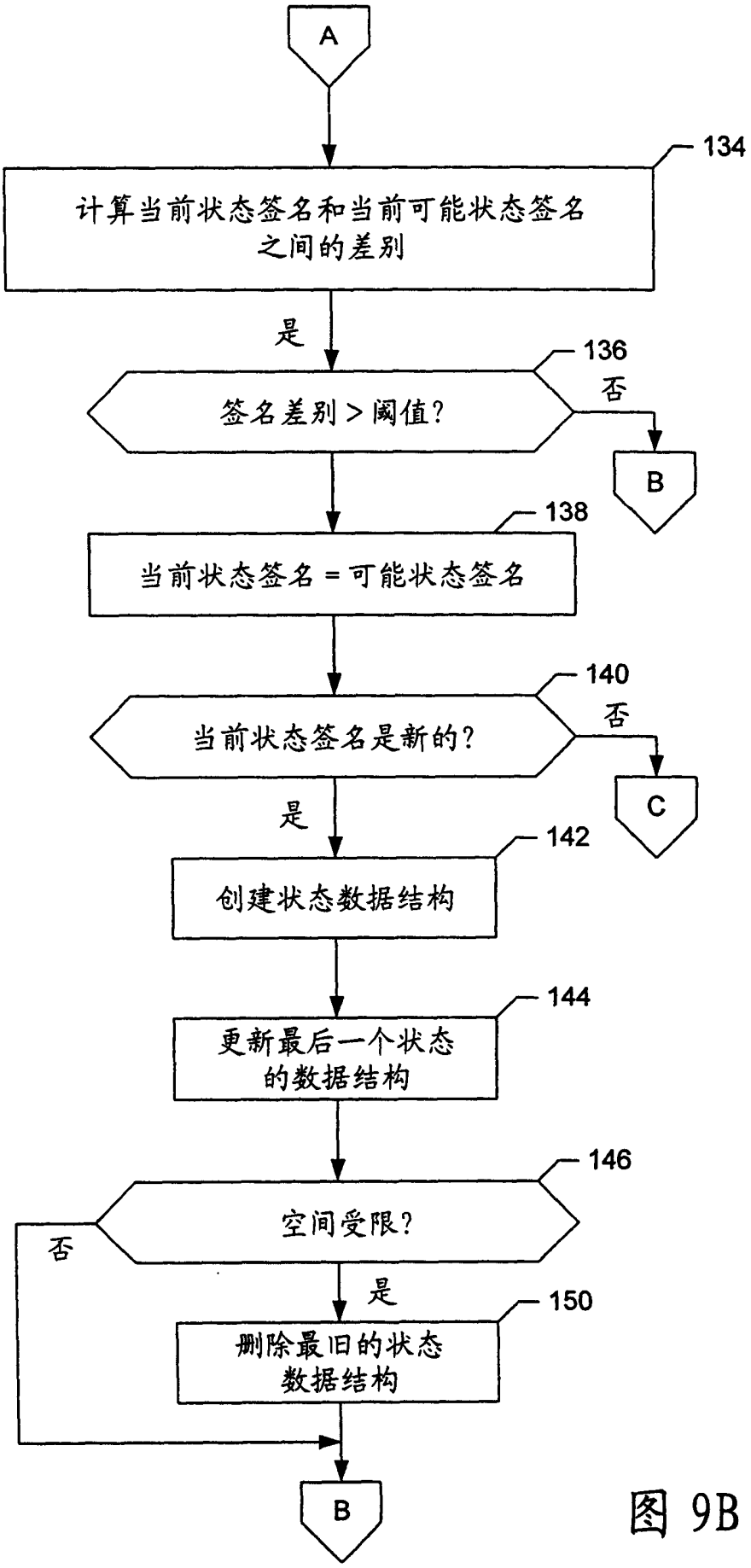

A
134
计算当前状态签名和当前可能状态签名之间的差别
是
136
签名差别 > 阈值?
否
B
138
当前状态签名 = 可能状态签名
140
当前状态签名是新的?
否
C
是
142
创建状态数据结构
144
更新最后一个状态的数据结构
146
空间受限?
否
是
150
删除最旧的状态数据结构
B

图 9B

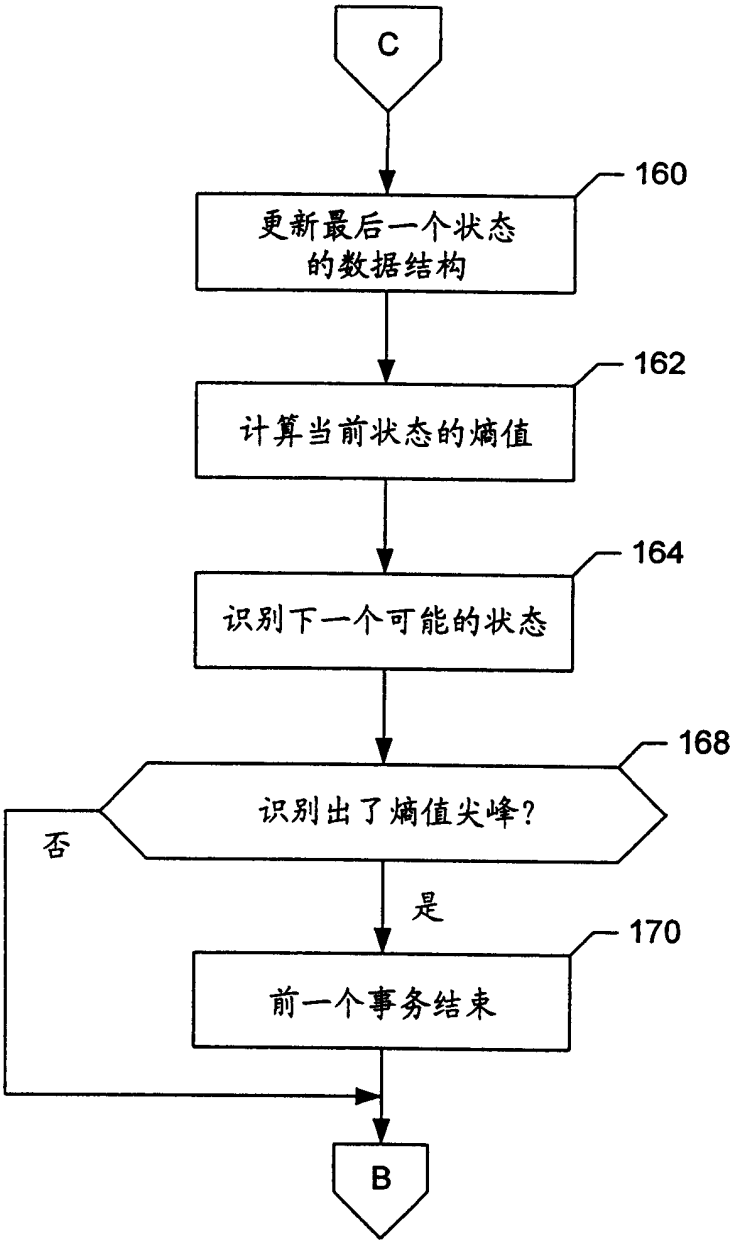

C
160
更新最后一个状态的数据结构
162
计算当前状态的熵值
164
识别下一个可能的状态
168
识别出了熵值尖峰?
否
是
170
前一个事务结束
B

图 9C

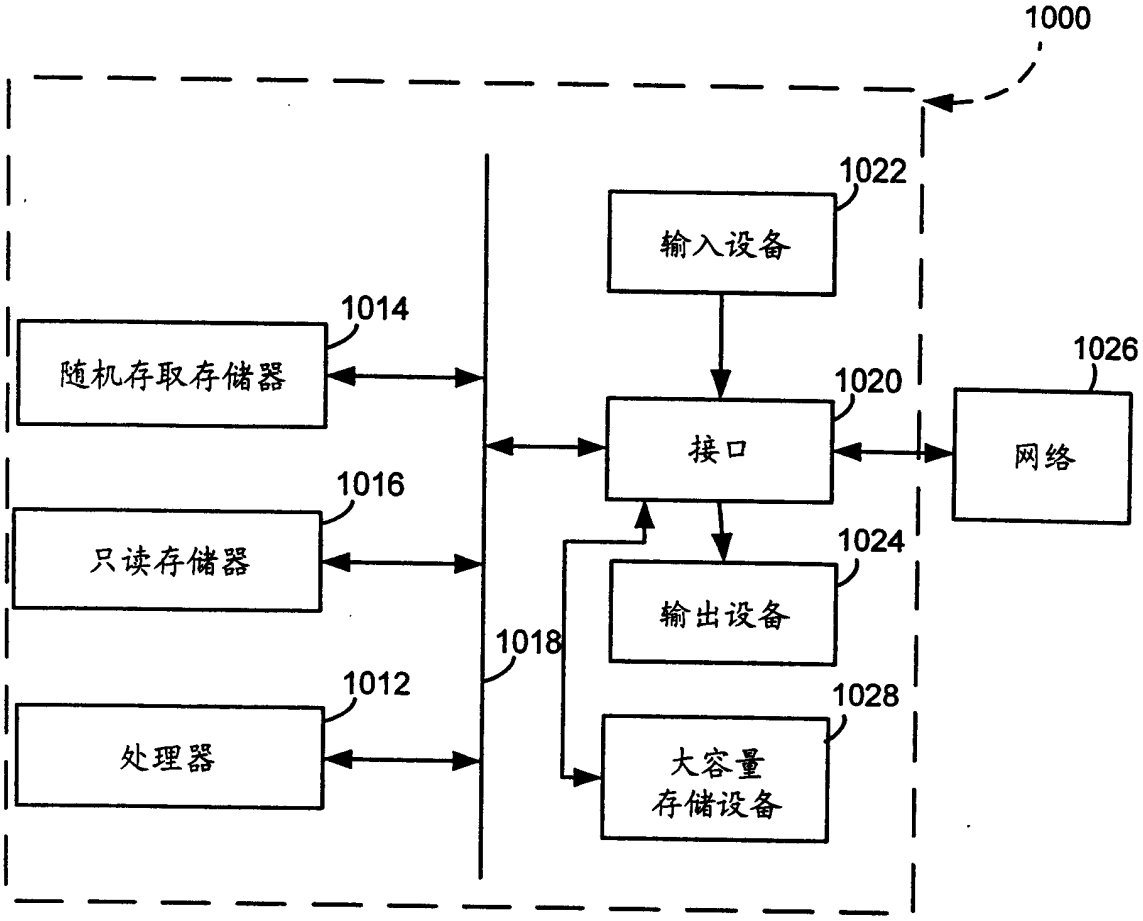

1000
1022
输入设备
1014
随机存取存储器
1026
1020
接口
网络
1016
只读存储器
1024
输出设备
1018
1012
处理器
1028
大容量
存储设备

图 10