[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/45

[12] 发 明 专 利 说 明 书

专利号 ZL 00117671.4

[45] 授权公告日 2005 年 10 月 26 日

[11] 授权公告号 CN 1224902C

[22] 申请日 2000.5.26 [21] 申请号 00117671.4
[30] 优先权
[32] 1999. 5.27 [33] US [31] 09/321,228
[71] 专利权人 太阳微系统公司
地址 美国加利福尼亚州
[72] 发明人 吉拉德·布拉查 梁 胜
蒂蒙西 G·林德霍尔姆
审查员 李延峰

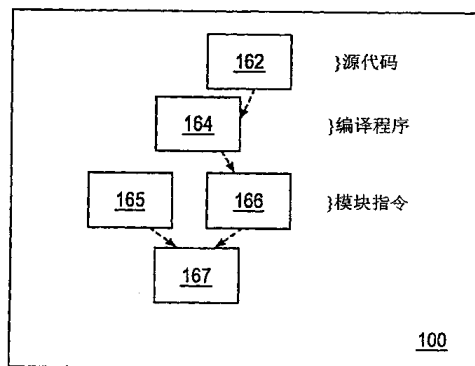
[74] 专利代理机构 中原信达知识产权代理有限责任公司
代理人 李 辉 谷慧敏

权利要求书 5 页 说明书 34 页 附图 18 页

[54] 发明名称 用于最低上界类型的符号计算的数据流算法

[57] 摘要

一种用于验证计算机程序的模块中的指令的方法，计算机程序，信号传输，装置和系统，其中该模块将要动态地链接到至少一个其它模块。 其首先确定，检验所装载的第一模块中的指令是否需要在不同于第一模块的一个或多个被引用模块中的至少两个被引用类型的最小上界(LUB)类型。 如果需要该信息，则写入用于该被引用模块的约束，而无需装载该被引用模块。 该约束的形式是“至少两个类的集合从一个指定类继承”。



ISSN 1008-4274

知识产权出版社出版

1. 一种用于验证计算机程序的模块中的指令的方法，该方法包括：

5 首先确定，检验第一模块中的指令是否需要在不同于第一模块的一个或多个被引用模块中定义的至少两个被引用类型的最小上界 类型；和

如果需要该最小上界类型，则写入一约束，而无需装载该一个或多个被引用模块的任何一个，其中所述约束相应于所述的一个或多个

10 被引用模块。

2. 根据权利要求 1 的方法，其中该约束的形式是“至少两个被引用类型的列表是一个指定类型的子类型”。

15 3. 根据权利要求 1 的方法，其中该约束的形式是“一个被引用类型是一个指定类型的子类型”，并且为该至少两个被引用类型中的每个被引用类型写入约束。

4. 一种用于装载计算机程序的模块的方法，该方法包括：

20 读取形式为“至少两个类型的列表是一个指定类型的子类型”的约束；

如果该指定类型和该至少两个类型的列表的至少一个类型在已经装载的模块中，则实施该约束；和

如果有的话，为尚未装载的模块中定义的该至少两个类型的列表

25 的每个类型写入一个新约束，该新约束的形式是“尚未装载的模块中定义的类型的新列表是一个指定类型的子类型”。

5. 根据权利要求 4 的方法，所述实施该约束的步骤进一步包括，如果该至少两个类型的列表的该至少一个类型不是该指定类型的子类

30 型，则发送一个错误。

6. 一种用于验证计算机程序的模块中的指令的验证程序装置，包括：

计算机可读存储介质，用于存储计算机程序的模块；

5 存储器，用于装载一模块；和

处理器，用于首先确定，检验第一模块中的指令是否需要在不同于第一模块的一个或多个被引用模块中的至少两个被引用类型的最小上界 类型；并且如果需要该最小上界类型，则写入一约束，而无需装载该一个或多个被引用模块的任何一个，其中所述约束相应于所述的

10 一个或多个被引用模块。

7. 一种用于计算机程序的模块的装载装置，包括：

计算机可读存储介质，用于存储计算机程序的模块；

存储器，用于装载一模块；和

15 处理器，用于从该存储介质和该存储器中的至少一个读取形式为“至少两个类型的列表是一个指定类型的子类型”的约束，从而如果该指定类型和该至少两个类型的集合的至少一个类型在已经装载的模块中，则实施该约束，并且如果有的话，为尚未装载的模块中定义的该至少两个类型的列表的每个类型写入一个新约束，

20 其中该新约束的形式是“尚未装载的模块中定义的类型的新列表是一个指定类型的子类型”。

8. 根据权利要求 7 的装置，其中处理器进一步用于，如果该至少两个类型的集合的至少一个类型不是该指定类型的子类型，则通过

25 发送一个错误来实施该约束。

9. 一种信号传输，包括：

通信线路上的载波；和

表示使用载波传送的计算机控制命令的信号，用于首先确定，检

30 验第一模块中的指令是否需要在不同于第一模块的一个或多个被引用

模块中定义的至少两个被引用类型的最小上界 类型；并且如果需要该信息，则写入一约束，而无需装载该一个或多个被引用模块的任何一个。

5 10. 一种信号传输，包括：

通信线路上的载波；和

表示使用载波传送的计算机控制命令的信号，用于读取形式为“至少两个类型的列表是一个指定类型的子类型”的约束，从而如果该指定类型和该至少两个类型的列表的至少一个类型在已经装载的模块

10 中，则实施该约束，并且如果有的话，为尚未装载的模块中定义的该至少两个类型的列表的每个类型写入一个新约束，

其中该新约束的形式是“尚未装载的模块中定义的类型的新列表是一个指定类型的子类型”。

15 11. 一种用于对计算机中的模块进行验证和装载的系统，该系统包括：

网络；

连接到该网络的计算机可读存储介质，用于存储计算机程序的模块；

20 连接到该网络的存储器，用于装载一模块；

连接到该网络的处理器，用于，首先确定检验第一模块中的指令是否需要不同于第一模块的一个或多个被引用模块中的至少两个被引用类型的最小上界 类型；并且如果需要该最小上界类型，则写入一约束，而无需装载该一个或多个被引用模块的任何一个，

25 连接到该网络的处理器，用于从该存储介质和该存储器中的至少一个读取形式为“至少两个类型的列表是一个指定类型的子类型”的约束，从而如果该指定类型和该至少两个类型的集合的至少一个类型在已经装载的模块中，则实施该约束，并且为尚未装载的模块中定义的该至少两个类型的列表的每个类型写入任何新的约束，其中该新约

30 束的形式是“尚未装载的模块中定义的类型的新列表是一个指定类型

的子类型”；

其中所述约束相应于所述的一个或多个被引用模块。

12. 一种用于验证计算机程序的模块中的指令的方法，该模块将要动态地链接到至少一个其它模块，该方法包括：

合并第一装载模块的至少两个类型快照；

确定第一快照中的一个固定位置是否保存一个尚未装载模块中定义的一被引用类型；

确定第二快照中的该固定位置是否保存一不同类型；和

如果两个确定的结果都为真，则将一个包括该被引用类型和该不同类型的列表置于一合并快照的该固定位置，

从而可以无需完全填充的类型点阵进行验证。

13. 一种用于验证计算机程序的模块中的指令的计算机程序产品，该模块将要动态地链接到至少一个其它模块，该计算机程序产品包括：

计算机可读存储介质；和

存储在存储介质上的计算机控制指令，用于合并第一装载模块的至少两个类型快照，确定第一快照中的一固定位置是否保存一尚未装载模块中定义的一被引用类型，确定第二快照中的该固定位置是否保存一不同类型，并且如果两个确定的结果都为真，则将包括该被引用类型和该不同类型的列表置于一合并快照的该固定位置，

从而可以无需完全填充的类型点阵进行验证。

14. 一种用于计算机程序的模块的验证装置，包括：

计算机可读存储介质，用于存储计算机程序的模块；

存储器，用于装载一模块；和

处理器，用于合并第一装载模块的至少两个类型快照，从而确定第一快照中的一固定位置是否保存一尚未装载模块中定义的一被引用类型，确定第二快照中的该固定位置是否保存一不同类型，并且如果

两个确定结果都为真，则将包括该被引用类型和该不同类型的列表置于一合并快照的该固定位置，

从而可以无需完全填充的类型点阵进行验证。

5 15. 一种信号传输，包括：

通信线路上的载波；和

表示使用载波传送的计算机控制命令的信号，用于合并第一装载模块的至少两个类型快照，确定第一快照中的一固定位置是否保存一尚未装载模块中定义的一被引用类型，确定第二快照中的该固定位置是否保存一不同类型，并且如果两个确定结果都为真，则将包括该被引用
10 类型和该不同类型的列表置于一合并快照的该固定位置，

从而可以无需完全填充的类型点阵进行验证。

15

用于最低上界类型的符号计算的数据流算法

5 本申请涉及 Yellin 和 Gosling 于 1995 年 12 月 20 日提交的，题为"具有数据类型限制和对象初始化的预验证的字节代码程序解释器装置和方法"的美国专利申请序号 No.575,291(P1000)，现在是美国专利 No.5,740,441；Bracha 和 Liang 于 1998 年 8 月 14 日提交的，题为"用于类型安全、迟缓、用户定义的类装载的方法和装置"的美国专利申请
10 序号 No.09/134,477(P3135)；在此作为一个整体引入这些公开作为参考。

本申请还涉及 1999 年 5 月 27 日提交的，题为"完全迟缓链接"的美国专利申请序号 No.09/321,223[50253-228](P3564)；1999 年 5 月 27
15 日提交的，题为"逐模块验证"的美国专利申请序号 No.09/320,574[50253-229](P3565)；1999 年 5 月 27 日提交的，题为"具有逐模块验证的完全迟缓链接"的美国专利申请序号 No.09/321,226[50253-230](P3566)；1999 年 5 月 27 日提交的，题为"为逐模块验证缓存非置信模块"的美国专利申请序号
20 No.09/320,581[50253-235](P3810)。

技术领域

本发明涉及计算机编程语言，特别是具有验证指令的动态链接同时支持迟缓装载的计算机编程语言。

25

背景技术

通常，以人们易于理解的高级语言将计算机程序编写为源代码语句。在实际执行计算机程序时，计算机响应由直接控制中央处理单元(CPU)操作的二进制信号构成的指令组成的机器码。使用被称为编译
30 程序的专用程序读取源代码并将其语句转换成具体 CPU 的机器码指令

在本领域中是熟知的。所产生的机器码指令是依赖于平台的，就是说，不同的计算机设备具有带有由不同机器码表示的不同指令集的不同CPU。

5 通过组合几个比较简单的程序构成更强大的程序在本领域中也是已知的。可通过在编译前将几段源代码复制在一起，然后编译组合的源代码来进行该组合。当频繁使用一段源代码语句而没有改变时，通常最好是由其自身将其编译一次，以产生一个模块，并且仅当实际需要该功能块时将该模块与其它模块组合。这种编译后的模块组合被称
10 为链接。刚好在执行之前，当依据运行时间状况决定组合哪些模块并且模块组合发生在运行时间时，该链接被称为动态链接。

链接的优点在于可一次一个模块地开发程序，并由于不同开发者能够在不同地点同时开发不同模块，因此能够提高生产率。

15

在运行时间进行的链接，即动态链接的优点在于不需要链接执行期间不使用的模块，从而减少了必须执行的工作量，并且很可能减少执行代码的规模。通常，在链接前必须装载模块，即识别并将其引入存储器。将模块链接延缓到直到需要该模块为止将允许延期装载这些
20 模块，并称其为迟缓装载。

当汇编几个已经独立编写的模块时，谨慎的作法是，检验在其自己的全部范围内正确地执行每个模块，即模块内检验，以及多个模块一起正确地工作，即模块间检验。从 JAVA[TM] 编程语言的设计者使
25 用的术语推论，这种后编译模块检验可被称为验证。

得益于动态链接的计算机体系结构的一个实例是如 Sun Microsystems，Inc 的 JAVA[TM] 虚拟机(JVM)之类的虚拟机(VM)，这是一种可用硬件或软件实现的抽象计算机体系结构。在下面的 VM 描
30 述中包括了规定的任何一种实现方案。

VM 可以以下面的方式提供平台独立性。把如 JAVA^{TM} 编程语言之类的高级计算语言中表示的语句编译成独立于系统的 VM 指令。VM 指令对 VM 的关系正如机器码对中央处理单元(CPU)的关系。然后可将 VM 指令从一台机器传送到另一台机器。每个不同的处理器需要其自己的 VM 实现。VM 通过一次一条或多条指令地对 VM 指令进行翻译或解释来运行 VM 指令。在许多实现中，该 VM 是在特定计算机的 CPU 上运行的程序，但 VM 指令也可用作特定处理器或设备的本机指令集。在后一种情况下，VM 是一台"实际"的机器。包括动态链接和验证的 VM 也可进行其它操作。

使用该 VM 的编程处理有两个与其相关联的时间纪元；"编译时间"是指将高级语言转换成 VM 指令的步骤，"运行时间"是指在 JAVA^{TM} VM 环境中解释指令以执行该模块的步骤。在编译时间与运行时间之间，从语句编译的指令模块可在延长的、任意时间周期内驻留待用，或可从一个存储装置传送到另一个存储装置，包括跨网络传送。

JAVA^{TM} 虚拟机的实例可说明实现具有验证和具有或没有迟缓装载的动态链接的尝试中所遇到的问题。JVM 是一种面向对象的 JAVA^{TM} 高级编程语言的特定 VM，如 T.Lindholm 和 Frank Yellin，Addison-Wesley，Menlo Park 于 1997 年在 California 的 JAVA^{TM} 虚拟机说明书中对常规 JVM 所描述的，这种 JAVA^{TM} 高级编程语言被设计用来执行动态链接、验证和迟缓装载。

面向对象的编程技术被广泛使用，例如 JAVA^{TM} 平台使用的那些面向对象编程技术。面向对象程序的基本单元是对象，其具有方法(过程)和域(数据)，在此被称为成员。将共享成员的对象分组成类。一个类定义了在该类中对象的共享成员。每个对象则是其所属类的一个特定实例。实际上，经常将一个类用作模板，以便产生具有相似特性的

多个对象(多个实例)。

这些类的一个特性是封装，描述了除非通过接口暴露外，使外部用户，和其它类无法察觉该类中的成员的实际实现的特性。这使得这些类适合于分布式开发，例如，由不同的开发者在网络上的不同地点开发。通过汇编所需的类，把它们链接在一起，并执行得到的程序可形成一个完整的程序。

类享有继承的特性。继承是能使一个类继承另一个类的所有成员的机理。从另一个类继承的类被称为子类；提供该属性的类是超类。可用符号将其写成子类<=超类，或超类=>子类。子类可通过增加附加成员来扩展超类的容量。子类可通过提供具有相同名称和类型的替代成员来超越超类的属性。

JVM 以用于编译的类的特定二进制格式，即类文件格式操作。类文件包含 JVM 指令和符号表，以及其它辅助信息。为安全起见，JVM 对类文件中的指令施加强格式和结构约束。在特定实例中，JVM 指令是类型专用的，意图在于对如下面说明的给定类型的操作数操作。可由任何 VM 施加同样的约束。设计类文件以代表以 JAVA™ 编程语言写成的程序，而且也支持几种其它编程语言。JVM 可作为具有可通过有效类文件表示的功能性的任何语言的宿主。

在类文件中，变量是已与一种类型相关联的存储位置，该类型有时被称为其编译时间类型，其是原始类型或引用类型。引用类型是到对象或是不指向对象的专用空引用的指针。子类的类型被称为其超类的子类型。JVM 的原始类型包括 boolean (取真值为真和假)，char(用于单一代码字符的代码)，byte(8 个带有符号的 0 或 1 比特)，short(带符号的短整数)，int(带符号的整数)，long(带符号的长整数)，float(单精度浮点数)或 double(双精度浮点数)。

类类型的成员是域和方法；这些包括从超类继承的成员。类文件还命名超类。成员可以是公用的，是指可由任何类的成员访问。专用成员仅可由包含其声明的类的成员访问。被保护的成员可由声明类的成员或来自其声明的封装中任何地方的成员访问。在 JAVA™ 编程语言中，可对类分组并可对组命名；命名的类组是一个封装。

JVM 的实际指令包含在由类文件编码的类的方法中。

当 JAVA™ 语言程序违反操作的约束时，JVM 检测一无效条件并将该错误作为异常发送到程序。将异常从其出现的点发出(throw)，并且在向其传送控制的点被捕捉。由类 Throwable 或其子类之一的实例表示每个异常，可使用这样一个对象从异常出现的点向该程序的部分传递信息，即向用于对其进行捕捉和处理的异常处理程序传递信息。

JVM 通过调用一些指定类的"main"方法，传送给它一个作为字符串数组的单个自变量来开始执行。这样使得指定类被装载、链接和初始化。

装载是指通常通过取出预先从源代码编译的二进制表示，找到具有特定名称的类或封装的二进制形式的过程。在 JVM 中，装载步骤取出表示所希望类的类文件格式的二进制类。由引导类装载程序或用户定义的类装载程序实现该装载过程。用户定义的类装载程序是由类本身定义的。类装载程序可表示要搜索的特定位置顺序，以便找到表示一命名的类的类文件。类装载程序可缓存类的二进制表示，根据预期的使用预取，或把一组相关的类装载在一起。预取的类或装载的组越多，装载程序越"急切"。“迟缓”装载程序对尽可能少的类进行预取或分组。常规的 JVM 规范允许在急切和近乎完全迟缓之间的宽频谱的装载行为。

如果仅在一个类首先需要执行当前处理的类的指令时，其调用一个类装载程序以便装载该类，则 VM 是完全迟缓。如果实现，完全迟缓装载不浪费运行时间资源，例如系统存储器及执行时间，装载在运行时间不严格需要的类。

5

JVM 中的链接是在存储器中取类的二进制形式并将其组合成 VM 的运行时间状态，以便可执行的过程。在链接类之前必须装载类。根据 JVM 规范，在链接中涉及到三个不同的活动:验证、准备和符号引用的决定。

10

在验证期间，检验类文件格式中对二进制类的必要约束。这是 JVM 安全规定的基础。验证确保了 JVM 不尝试可导致无意义结果或可损害操作系统、文件系统、或 JVM 本身的完整性的违法操作。然而，检验这些约束有时需要了解其它类中的子类型关系，因此，成功的验证通常取决于由被验证的类引用的其它类的特性。这样具有使当前的用于验证的 JVM 设计规范是上下文有关的效果。

15

JVM 的二进制类实质上是一般程序模块的例子，该程序模块包含从编译的源语句产生的指令。有效性检验的上下文有关性意味着这些检验取决于跨越多于一个模块分散的信息，即，在此这些检验被称为跨模块检验或模块间检验。不需要来自另一个模块的信息的有效性检验在此被称为模块内检验。

20

上下文有关的验证有一些缺陷。例如，在象 JAVA™ 平台这样的面向对象编程系统中，当验证程序需要检验未装载的类间的子类型关系时，它导致验证程序启动类装载。即使从未执行引用其它类的代码，该装载也可发生。就是说，上下文有关的验证可干扰完全迟缓装载。由于该原因，与除了实际执行的指令引用某些类以外都不装载这些类的过程相比，装载可能消耗存储器并在运行时间减慢执行。

25

30

当验证是上下文有关时，同样没有在运行时间之前一次验证一个类或模块的规定。由于不能预先，例如在运行时间前验证类，这是一个缺陷，因此验证肯定招致花费运行时间。因此，需要在运行时间前进行逐模块，也称为一次一模块验证。在此将该验证称为预验证，因为从技术上讲，它与 JVM 链接运行时间期间发生的验证不同。

另外，由于在运行时间进行验证，每当装载一个类时，已运行一次，并通过验证的类再次受到验证，即使在相同的主计算机上的相同应用中使用该类，其中很可能没有新的验证问题，或是可安排一种情况，以致不能做出影响验证的变化。这样会导致冗余验证，从而在运行时间期间需要比应需要的更多的存储器并且更慢地执行。因此需要一个使用预验证模块的选项，在运行时间不用进一步验证，或使验证最少。

对预验证和完全迟缓装载的需求是可能被分别满足的两种分离的需求。还需要同时支持逐模块预验证和完全迟缓装载。

对包括减少运行时间验证的预验证的需求可能与安全的目的相冲突，安全目的需要在运行时间检验提供给虚拟机或任何计算体系结构的所有模块，以防止非法或损坏操作。例如，在非置信的情况下，例如从因特网下载模块和其预验证输出，攻击者可能对预验证输出进行电子欺骗，可能使恶意的类表现为良性。因此，需要可在非置信的情况下使用的预验证，如跨越因特网下载模块。

对完全迟缓装载或逐模块预验证的需求产生了对类型点阵（type lattice）的替换表示的需求。类型点阵是表示类型间子类型关系的数学结构。由 JVM 建立类型点阵的表示，用于表示运行时间期间类的类型和子类型。JVM 还保持链接的类的所有属性的引用和类型。期望类似的运行时间结构用于任何动态链接过程。为支持逐类预验证或完全迟缓装载，必须在不完全了解类型点阵的情况下进行类型检验，另

外，大部分类型点阵通常在可能仍不需要装载的其它模块中定义。特别是，JVM 通常需要在验证期间找到类型点阵中的 LUB(最低上界)类型。因此，即使在不能使用类型点阵时，仍需要执行依赖于 LUB 的功能。

5

从下面结合附图对本发明的详细描述中将使本发明上面和其它特性，方面和优点变得显而易见。

发明内容

10 本发明的一个目的是支持链接期间的验证同时提供完全迟缓装载。对于动态链接程序，特别是 JVM，其优点是在指令(例如方法的)执行期间需要在专用的，定义的点迟缓地进行被引用模块(例如类)的所有决定。这些优点包括:

- 一次写入，任何时间运行(WORA)特性被改善。程序相对于链
15 接错误的行为对所有平台和实现是相同的。
- 大大地改善了可测试性。例如，不需要预先考虑可能链接类或方法的所有位置，并在不能找到类或方法的情况下尝试捕捉所有那些位置处的异常。
- 用户可以用一种可靠和简单的方式确定模块的存在。例如，通
20 过把那些引用放置在除可使用不同版本外不能执行的程序支路上，用户可以避免因调用在不同版本的运行时间环境中遗漏的模块造成的链接错误。

常规 JVM 规范的装载行为的广度不提供这些优点。

25

本发明的另一个目的是提供一次一模块的预验证。本发明的再一个目的是使用预验证的指令，以减少运行时间验证。某些 JAVA™ 平台的用户想对某些类执行上下文不相关的，或上下文独立的，验证检验。可在编译期间或之后和运行时间之前进行的上下文独立的检验有
30 许多优点。这些优点包括:

• 在运行时间之前可检测某些验证错误；

• 由于其所包含的验证代码量减少，如果仍需要的话，运行时间的链接部分更小并且更简单；和

• 用户可在逐模块的基础上，而不是逐应用的基础上存储模块(在安全储存库中，例如关系数据库管理系统)，并在运行时间前进行尽可能多的工作。这样避免了冗余验证并减少或消除了验证花费的运行时间。

本发明的再一个目的允许与可许可完全迟缓装载的运行时间验证组合的一次一模块(或类)预验证，以便同时享有二者的优点。

本发明的再一个目的允许来自将要验证的非置信源的模块，以提高应用预验证优点的适应范围。

本发明的另一个目的是在缺乏对类型点阵的全面了解时使用 LUB 的替代物，以简化模块间的有效性检验。

本发明的这些和其它目的及优点是由一种方法、计算机程序、信号传输和装置提供的，用于验证要与至少一个其它模块动态链接的计算机模块中的指令。其首先确定检验装载的第一模块中的指令是否需要与第一模块不同的一个或多个被引用模块中的至少两个被引用类的最小上界（LUB）类。如果需要该信息，写入用于被引用模块的约束，而不需要装载被引用模块。该约束的形式是“该至少两个类的集合从一指定类继承”。

在本发明的另一方面中，一种方法、计算机程序、信号传输和装置验证要与至少一个其它模块动态链接的计算机模块中的指令。读取形式为“至少两个类的集合从一指定类继承”的约束。如果该指定类和该其它两个类中的至少一个在已经装载的模块中，则实施该约束。如果有的话，为属于尚未装载的模块的其它类中的每一个写入新约

束。该新约束的形式是“未装载模块的每个类从指定类继承”。

在本发明另一个方面，动态链接和装载包括网络和连接到该网络用于存储计算机程序模块的计算机可读存储介质。一个可装载模块的存储器也连接到该网络。配置连接到网络的处理器，以便首先确定检验装载的第一模块中的指令是否需要与第一模块不同的一个或多个被引用模块中至少两个被引用类的最小上界（LUB）类。如果需要该信息，则写入用于该被引用模块的约束，而无需装载该被引用模块，其中该约束的形式是“该至少两个类的集合从一指定类继承”。连接到网络的相同或不同处理器被配置为，从存储介质和存储器中的至少一个读取形式为“至少两个类的集合从一指定类继承”的约束。如果该指定类和该其它两个类中的至少一个已经在装载的模块中，则实施该约束。如果有的话，为未装载的模块的每个类写入新约束。该新约束的形式是“未装载模块的每个类从指定类继承”。

本发明进一步涉及一种信号传输，包括：通信线路上的载波；和表示使用载波传送的计算机控制命令的信号，用于首先确定，检验第一模块中的指令是否需要在不同于第一模块的一个或多个被引用模块中定义的至少两个被引用类型的最小上界类型；并且如果需要该信息，则写入一约束，而无需装载该一个或多个被引用模块的任何一个。

本发明还提供了一种信号传输，包括：通信线路上的载波；和表示使用载波传送的计算机控制命令的信号，用于读取形式为“至少两个类型的列表是一个指定类型的子类型”的约束，从而如果该指定类型和该至少两个类型的列表的至少一个类型在已经装载的模块中，则实施该约束，并且如果有的话，为尚未装载的模块中定义的该至少两个类型的列表的每个类型写入一个新约束，其中该新约束的形式是“尚未装载的模块中定义的类型的新列表是一个指定类型的子类型”。

本发明又提供了一种信号传输，包括：通信线路上的载波；和表

示使用载波传送的计算机控制命令的信号，用于合并第一装载模块的至少两个类型快照，确定第一快照中的一固定位置是否保存一尚未装载模块中定义的一被引用类型，确定第二快照中的该固定位置是否保存一不同类型，并且如果两个确定结果都为真，则将包括该被引用类型和该不同类型的列表置于一合并快照的该固定位置，从而可以无需完全填充的类型点阵进行验证。

附图说明

本发明系统的目的，特性和优点从下面的描述中是显而易见，其中:

图 1A 是适合于执行本发明使用的计算机系统示例的示意图。

图 1B 是图 1A 的计算机的硬件配置示例的方框图。

图 1C 示出了适合于存储根据本发明的程序和数据信息的存储介质的示例。

图 1D 是适合于承载根据本发明的数据和程序的网络体系结构的方框图。

图 1E 是根据本发明配置的计算机的方框图。

图 2 是在与 JAVA™ 编程语言类似的伪语言中具有方法 FOO 和引用类 A 和 B 的类 BAR 的示例。

图 3 是描绘来自图 2 的类 BAR 示例的完全急切装载的流程图。

图 4A 是来自图 2 的类 BAR 示例的近乎迟缓装载的流程图。

图 4B 是描绘针对图 4A 中描绘的近乎迟缓装载的步骤 475，在新近更新的 JVM 中采用的访问类型检验的流程图。

图 5A 是描绘在针对图 2 的类 BAR 示例的图 4A 的链接步骤 435 中的验证的流程图。

图 5B 是描绘针对来自图 2 的类 BAR 示例，在来自图 5A 的步骤 530 的一个实施例期间的验证方法的流程图。

图 5C 是描绘在图 5B 的验证指令步骤 537 中的指令验证的流程图。

图 6A 是描绘针对来自允许图 2 的完全迟缓装载的类 BAR 示例，

对于来自图 5A 的步骤 530，在本发明的实施例期间的验证方法的流程图。

图 6B 是描绘根据本发明的实施例，在图 6A 的验证指令步骤 637 中验证指令的流程图。

5 图 6C 是描绘针对图 2 的类 BAR 示例，在图 4A 的步骤 475 期间根据本发明实施例的验证约束检验的流程图。

图 7A 是描绘根据本发明针对来自图 2 的类 BAR 示例一次一类预验证的流程图。

图 7B 是描绘在图 7A 的步骤 716 期间预验证方法的流程图。

10 图 7C 是描绘根据本发明一个实施例，在来自图 2 的类 BAR 示例的运行时间验证期间，在图 5A 中的步骤 530 期间使用的逐类预验证的流程图。

图 8 是描绘根据本发明的另一个实施例，在来自图 5A 的步骤 530 期间使用逐类预验证的流程图，允许具有来自图 2 的类 BAR 示例的

15 逐类预验证的完全迟缓装载。

图 9 是根据本发明的另一个实施例为具有用于可信的类和验证约束的缓存的预验证配置的计算机的流程图。

标注和术语

20 下面给出关于可在计算机或计算机网络上执行的程序过程的详细描述。这些程序描述和表示法是本领域技术人员使用的含义，以便最有效地将其工作要点传达给本领域的其他技术人员。

在此，通常将该过程设想为导致所希望结果的独立的步骤序列。

25 这些步骤是需要物理量的物理操作的那些步骤。通常，虽然不是一定的，这些量采用能够存储，传送，组合，比较，以及操作的电或磁信号的形式。主要是为了共同使用的缘故，实践证明将这些信号看作是比特，值，元素，符号，字符，项，数字等是很方便的。然而，应该指出，所有这些和相似项与适当的物理量相关联，并且仅是便于应用

30 于那些量的标记。

另外，所进行操作经常是指术语，例如相加或比较，这些术语通常与由操作人员进行的智力操作相关联。在大多数情况下，在形成本发明一部分的，在此描述的任何操作中，操作人员不必或不需要具有这种能力；该操作是机器操作。用于执行本发明的操作的有效机器包括通用数字计算机或类似设备。

本发明还涉及执行这些操作的装置。可以专为所需的目的构成该装置，或是其可包括由计算机中存储的计算机程序有选择地驱动或重新构成的通用计算机。在此给出的过程不固定涉及特定计算机或其它装置。各种通用机器可与根据在此讲述写入的程序一起使用，或者其可证明便于构成执行所需方法步骤的更专用的装置。这些机器中的各种所需结构从给出的描述中是显而易见的。

优选实施方式

图 1A 示出了一个适合于实现本发明的类型的计算机。在图 1A 中的外部视图看，计算机系统具有一个带有盘驱动器 110A 和 110B 的中央处理单元 100。盘驱动器指示 110A 和 110B 仅仅是多个可为计算机系统容纳的盘驱动器的符号。这些盘驱动器一般包括一个软盘驱动器，例如 110A，一个硬盘驱动器（未在外部示出），和槽 110B 代表的 CD ROM 或 DVD 驱动器。驱动器的数量和类型一般随不同计算机配置而改变。计算机有一个显示信息的显示器 120。还有可用作输入设备的键盘 130 和鼠标器 140。图 1A 中示出的计算机可以是 Sun Microsystems, Inc.生产的 SPARC 工作站。

图 1B 示出了图 1A 的计算机的内部硬件的方框图。总线 150 用作互连其它计算机组件的主信息干线。CPU 155 是系统的中央处理单元，进行执行程序所需的计算和逻辑操作。只读存储器（160）和随机存取存储器（165）构成了计算机的主存储器。盘控制器 170 把一个或多个盘驱动器连接到系统总线 150。这些盘驱动器可以是软盘驱

动器，例如 173，内置或外置硬盘驱动器，例如 172，或 CD ROM 或 DVD（数字视盘）驱动器，例如 171。显示器接口 125 连接显示器 120，并使来自总线的信息显示在显示器上。可以通过通信端口 175 与外部设备通信。

5

图 1C 示出了图 1B 中 173 或图 1A 中的 110A 可以使用的示例存储介质。诸如软盘，或 CD-ROM，或数字视盘之类的存储介质一般包含用于控制计算机的程序信息，以使计算机能够执行根据本发明的功能。

10

图 1D 是适用于承载根据本发明的一些方面的数据和程序的网络构造的方框图。网络 190 用作使客户计算机 100 与一个或多个服务器，例如服务器 195，连接，以下载程序和数据信息。客户机 100’也可以经过网络服务提供商，例如 ISP180，连接到网络 190。如以后将说明的，可以通过网络配置以硬件或软件实现的有关虚拟机（VM）的

15

单元或其它计算体系结构。

图 1E 示出了一个配置有有关虚拟机组件的单一计算机。组件包括：在计算机中的存储介质的一个或多个逻辑块中的源代码语句 162，

20

一个编译源代码 162 以产生一个或多个包含 VM 指令之类指令的模块 165，166 的编译程序 164，和一个像虚拟机（VM）167 那样的把一个或多个模块 165，166 作为输入并执行它们产生的程序的处理器。尽管在图 1E 示出了在一个计算机上的组件，但是应当知道模块 165 和例如 VM 167 这样的处理器需要至少临时地存在于相同的计算机上。

25

可以从一个运行编译程序以便从源代码产生模块的不同计算机发送模块。例如，图 1D 示出了服务器 195 上的一个编译程序 194 和源代码 192，和两个不同的虚拟机实现 150，151，两个客户机 100，100’分别各有一个。源代码 192（和图 1E 中的 162）可以是任何语言，但是优选是 JAVA™ 语言编程语言，并且可以是由人类程序员编写或从另

30

外一个程序输出。可以通过网络 190 传送服务器 195 上的编译程序 194

产生的模块 196，并作为模块，例如 156，存储在一个客户机上，例如客户机 100。在那里，例如 VM 150 的特定平台实现可以执行模块 156 中的指令。

5 要特别指出，尽管本发明是用 JVM 说明的，但并不限于 JVM。本发明可以应用于任何在运行期间链接来自各种源的程序模块并且在执行它们之前验证这些程序模块的处理。

图 2 示出了用类似于 JAVA™ 编程语言的编程语言写出伪源代码，作为一个程序模块的伪源代码的例子，其代表一个表现了引起本发明要解决的问题的条件的类。第一行命名了类“BAR”。第一组省略号代表给予类 BAR 定义的其它语句，但在这里不予考虑。接下来一行到本例的结尾定义了类 BAR 中命名为 FOO 的方法（也表示为 BAR.FOO）；类型“void”指示在方法 FOO 的调用终止时不返回值。下面一行引入了一个在执行期间提供两个支路的“if else”结构。如果方法自变量，即“arg”是真，那么执行由下一组省略号、括号中的赋值语句和紧跟着的省略号代表的一个支路。赋值语句表示给命名为“var”的类 A 的变量赋予一个类 B 的新实例。因此，在这个支路中，对两个其它类进行引用，这两个类是被引用类，类 A 和类 B。下一行，if else 结构的 else，表示方法的另一个支路的开始，如果 arg 是假，那么进入这个支路。该另一个支路包括在下一对括号之间，并且由另一组省略号代表，指示不在本支路中对类 A 或 B 进行引用。两个支路最后再次汇聚到给变量 z 的值赋予其原始值的平方的语句。

利用示例类 BAR 及其方法 FOO，可以在 JVM 之类的虚拟机中显示出急切装载（eager loading），近乎迟缓装载（almost lazy loading），和完全迟缓装载（fully lazy loading）之间的差别以及本发明的优点。当然，JVM 并不是以图 2 中列出的类似 JAVA™ 编程语言操作，而是用一般由编译程序产生的包含指令的模块操作；编译程序以图 2 中所示的高级编程语言代码操作。

图 3 示出了由 JVM 进行的示例类 BAR 的完全急切装载。假设还没有装载类 BAR，当达到调用类 BAR 中定义的方法 FOO 的时刻，在步骤 310，JVM 利用 BAR 的类装载程序，例如装载程序 L1，把类 BAR 从某个存储装置装载到存储器中。然后，类 BAR 成为当前类。由于当前类引用类 A 和 B，如果它们还没有装载，那么在步骤 320，急切 JVM 也调用装载程序来装载这两个类。在图 3 中，类 A 和 B 的类装载程序分别表示为 L2 和 L3；但是 L1，L2 和 L3 可以是相同的内建或用户定义的类装载程序，或者可以是任意两个相同，或可以每个都是不同的。

在链接期间 335，JVM 执行验证。前面参考的第 5，740，441 号美国专利中描述了验证期间使用的过程的许多细节。如上述专利中所描述的，验证包括识别一个试图处理错误类型的数据的方法中的任何指令序列，或任何会引起虚拟机的操作数堆栈下溢或溢出的指令。JVM 中的指令是特定类型的，因此通过指令操作的操作数必须与指令定义的类型匹配。操作数堆栈溢出是试图把一个会造成堆栈超过类文件中定义的堆栈的预定最大容量的项目，例如原始值或对象引用，放入操作数堆栈，也就是说在堆栈已经装满时放入。当在堆栈上没有有效项目留下时，即，在堆栈已经清空时，指令试图从操作数堆栈取出项目时发生操作数堆栈下溢。可以预期，验证中可以包括能够在执行模块中指令之前进行的任何有效性检验。

如果模块验证失败，那么虚拟机识别出错误，并且不执行模块中的指令。在 JVM 的情况下，JVM 发出一个可以用类异常处理程序得体处理的链接或验证错误消息（未示出）。

如果模块验证成功并且完成了链接，那么执行可以开始。在本例的情况下，当 JVM 解释每个指令并执行之的时候，可以在步骤 340 初始化当前类 BAR，和在步骤 350 运行当前类的方法 FOO.BAR。解

释程序不需要检验类型，或操作数堆栈下溢或溢出，因为已经通过在链接期间 335 执行的验证完成了检验。

上述涉及动态链接的过程的两个优点是：可以安全地使用由他人开发和编译的类，并且在链接之后，执行速度更快。可以使用他人编译的类是因为为了防止无效和可能的危险操作，在执行前的链接期间验证过它们。由于在验证期间进行了类型检验和操作数堆栈溢出和下溢，在指令执行时不进行它们，所以执行时间更快。同样，可以安全地在执行中跳过在验证期间进行过的其它有效性检验。

在迟缓装载中，如图 4A 中所示，直到在执行中需要它时才装载类。其优点可以用图 2 中的样本类 BAR 说明。如果 arg 是假，那么不进行“if”支路中引用类 A 和 B 的赋值，并且不需要装载或链接 A 或 B。因此，利用迟缓装载在运行时使得处理更快。

例如，如图 4A 中所示，在步骤 410 中利用类装载程序 L1 装载了类 BAR 之后，不立即装载 BAR 引用的类 A 和 B。而是在步骤 435 在链接期间验证类 BAR；并且如果类 BAR 通过验证和链接，在步骤 440，JVM 继续初始化类 BAR。另一方面，如果类 BAR 没有通过链接和验证，那么发出一个错误消息（未示出）并且不试图执行（未示出）。在步骤 440 中初始化类 BAR 之后，在步骤 450 中执行类 BAR 中的主方法（main method）并最终调用方法 FOO。如果变量 arg 是假，那么在方法 FOO 中采用“else”支路，并且既不使用类 A 也不使用类 B。这在图 4A 中用决策步骤 460 代表，在步骤 460 中确定当前指令是否需要决定对类 B 的引用。如果不需要类 B，那么执行当前指令并且返回到 460 继续执行下一个指令，直到没有剩下要验证的指令为止。另一方面，如果变量 arg 是真，那么执行“if”支路。这个支路包括赋值，在赋值中把类 A 类型的变量 var 设定为一个类 B 的新实例。当遇到第一个引用类 B 的指令时，必须调用类 B 的一个方法（B 的构造函数），并且必须决定对类 B 的引用。步骤 460 代表的，问询是否必须

为这个指令决定 B 的测试被给予肯定的回答。那么，如果还没有装载类 B，步骤 470 利用类装载程序 L3 装载类 B。

在现有的 JVM 中，处理只是在图 4A 中所示的后装载步骤 475 中继续，并且直接前进到步骤 480。由于建立了类 B 的一个新实例，必须首先链接和初始化类 B。因此，如果还没有链接类 B，那么在下一步在步骤 480 中链接类 B。如果在步骤 480 中类 B 通过链接（包括验证），那么在步骤 490 初始化类 B，然后该处理继续到步骤 498，其中当前指令可以使用新决定的类 B。

这个流程表现为完全迟缓装载，其中直到需要在执行期间决定引用时才装载一个类。但是，如以后将要说明的，根据现有 JVM 规定，在链接期间 435 验证步骤可能需要类 B 的装载。在这种场合，不能认为该处理是完全迟缓，并且把该处理称为近乎迟缓装载。

图 4A 中所示近乎迟缓装载期间识别的一个问题是类名含糊不清。当一起编译几个类时，编译程序产生一个名称空间，名称空间包含有在名称空间内唯一的类名。但是，当不同编译程序在不同时间编译几个类时，不可能保证类的名称唯一性。在运行时间，类装载程序可能引入多个名称空间。结果，运行时间期间一个类的类型不是只通过其名称定义，而是通过类名和它的定义类装载程序，例如<BAR，L1>，的组合定义。这种环境可能欺骗验证程序，即使是在验证步骤装载决定类型所需的全部被引用类的现有系统中。在包括验证的链接 435 期间，假设被引用类，例如 B，具有当前类装载程序，例如 L1 要参看的类型，也就是说，假设类 B 的“类型”是<B，L1>。如果这种假设不是真，那么可能会产生访问特权的问题。例如，如果 B 的类装载程序 L3 与 BAR 的类装载程序 L1 不同，并且如果<B，L3>把<B，L1>声明为共用变量的一个变量声明为专用变量，那么 VM 可以允许从类 B 的外部访问该专用变量，并且可能危及程序的安全性。

在 1999 年 4 月发布的 JVM 规定的最新版本第二版中，如另一个有关专利申请中所述的那样避免这一问题，这个有关专利申请是 Bracha 和 Liang 等申请的名称为“用于类型安全，迟缓，用户定义类装载的方法和装置”的序列号为 09/134，477 号美国专利申请，这一专利申请也作为上述规定的参考。图 4B 示出了一个说明在 JVM 规定的第二版中使用的解决方法。利用这个解决方法，在后装载步骤 475 中包括进额外的步骤。步骤 473 确定在用 L3 实际装载类 B 时是否产生根据名称和 BAR 的装载程序 L1 假设的类型；即，步骤 473 确定<B，L3>是否等于<B，L1>。如果装载 B 实际产生了与假设类型不同的类型，那么类 B 不能通过名称/类型约束，并且在步骤 474 发出一个错误。否则，在步骤 479 继续执行。如下所述，上述专利申请中说明的方法并没有改变的事实是，步骤 435 中的链接可能需要装载被引用类 A 和/或 B 以检验子类型（subtyping），从而由类 BAR 使用它们。因此，上述的专利申请没有解决干扰提供完全迟缓装载的问题。

用图 5A，5B 和 5C 举例说明图 4A 的链接 435 内的验证步骤。图 5A 是一个流程图，示出了步骤 435 中链接类 BAR 包括开始验证当前类 BAR 的步骤 510，紧接其后的步骤 530，在步骤 530 中当前类 BAR 的方法 FOO 受到验证。接下来，在步骤 590 结束步骤 435 中的类 BAR 的验证。图 5B 中示出了步骤 530 的惯用实施 530a 期间使用的验证类 BAR 的方法 FOO 的过程。该方法在步骤 532 开始。如果该方法引用 A 和 B 之类的其它类，并且这些类尚未装载，那么验证过程可能需要装载类 A 和/或 B。在步骤 555 中对每个指令进行这个第一确定。如果需要被引用类 B，那么在步骤 540 确定类 B 是否已经装载。如果需要并且尚未装载，那么在步骤 550 装载该被引用类。因此，即使希望迟缓装载，方法验证可能在执行期间实际需要其它类之前装载了这些类。如步骤 537 表示的，如果在验证过程中发现 A 和 B 之间的不正确的子类型关系或其它验证问题，那么在步骤 539 发出一个验证错误。如果当前指令通过验证，那么在步骤 582 把验证过程继续到方法的结束，并返回到步骤 555，直到没有指令需要验证。也就是说，验证了

足够的指令集，因而可以开始方法的执行。

图 5C 示出了可以在步骤 537 进行的验证的某些示例细节；例如，在 JVM 中进行的和图 4B 的美国专利 5，740，441 中所述的。在这个示例中，保存每个指令的类型快照（snapshot）。类型快照是一种保持一系列的局部变量和一个操作数堆栈上的每个位置的类型的结构。当需要验证一个指令时，在步骤 533 装载它的快照。在步骤 534，确定指令的执行对快照中类型的影响。当一个指令是多个其它指令的后继时，例如当操作的两个支路汇聚时（如在图 2 中的示例方法 BAR.FOO 中的涉及“z”的语句处），必须把一个指令的类型快照与每个支路上的前驱指令的快照合并。这是在验证一个指令的同时通过在步骤 535 中确定当前指令的后继指令和在步骤 536 中把当前指令的快照与每个后继指令的快照合并完成的。如果快照中相同位置的原始类型不一致，这种合并将检测到一个验证的失败。该合并还用产生的合并快照中的最特殊的公用超类型（也称为最小上界 LUB，或最小公用超类型 LCS）替换进行合并的快照中的相同位置上的引用；如果该操作不能完成则验证失败。

如图 5B 所示，引用其它类的方法的验证可能妨碍完全迟缓装载，因为验证程序可能必须装载被引用类。被引用类的装载是不确定的——它取决于引用所处的上下文关系。在本示例中，为了使赋值语句合法，A 必须是 B 的超类。可以通过装载类 B 检验这种子类型关系。如果 A 是 B 的超类，那么 A 将在装载 B 的过程中装载其自身。如果在装载 B 之后没有装载 A，它不可能是超类 B。如果装载了 A，那么可以直接检验是否保持了这种关系。

带有完全迟缓装载的验证

为了完成根据本发明的带有验证的完全迟缓装载，必须能够延迟触发根据现有技术的装载的跨模块关系的检验。

如果在示例类 BAR 的验证期间 B 已经装载，那么可以立即确定是否保持了子类型关系。在装载 B 时超类型已经装载，不然它不可能是 B 的超类型。如果类 B 尚未装载，根据本发明，要给类 B 设置一个约束，在装载了类 B 之后将检验或实施这个约束。图 6A 至图 6C 中示出了本发明的这个实施例的一个示例。

图 6A 是图 5A 中所示方法验证步骤 530 的实现 530b 的流程图，并且是现有 JVM 规定的图 5B 中所示版本 530a 的替代。根据本发明的这个实施例，在步骤 632 开始方法验证，并在步骤 655 确定是否需要有关通常触发装载的类型的被引用类 B 的信息。如果被引用类 B 将传统地触发装载，程序接下来在步骤 640 检验是否已经装载了参考类 B。如果已经装载了类 B，那么在步骤 637 可以如现有 JVM 中那样继续处理，检验子类型和其它有效性检验，并且如果一个指令验证失败则在步骤 639 发出一个错误。如果指令确实满足包括模块间检验的有效性检验，那么在步骤 682 继续方法 FOO 的验证，循环通过足够的指令，直到没有为了开始执行而需要验证的指令。如果任何开始执行所需的指令验证失败，那么不开始模块的执行。

但是，如果在步骤 640 中确定尚未装载类 B，那么在此时不调用类 B 的类装载程序来装载类 B。而是在步骤 650 写入用于类 BAR 使用类 B 的验证约束。然后，假设对 B 的诸如子类型检验之类的所有跨模块检验通过，并且像以前那样在步骤 682 继续方法 FOO 的验证。在步骤 650 中导致写入约束的条件，和这些约束的形式将在以后对这些示例进行更详细的说明。根据本发明的这个实施例，如所希望的那样，链接期间的验证步骤 530b 不干扰模块的完全迟缓装载。

<u>带有 LUB 符号计算的验证</u>

如果尚未装载被引用模块或类，那么在这种完全迟缓链接期间不装载。在 JVM 中，这影响合并快照的结果，因为可能不知道在图 5C 中所示的步骤 538 中插入的 LUB。用另一种方式来说，JVM 不能给

出充分填充的类型点阵的代表来确定多个不同被引用类型的 LUB。图 6B 示出了如何根据本发明的一个实施例完成合并快照功能的。步骤 633，634，635 和 636 与对应的步骤 533，534，535 和 536 类似。但是，在步骤 638 中，如果不知道 LUB，那么不能把它的类型插入到合并快照的适当位置。而是插入前驱指令快照中适当固定位置上的被引用类型的一个列表，或是与合并快照中的该固定位置关联。也就是说，不是通过装载构造类型点阵所需的类或模块来识别和把对快照中的 LUB 的引用放在这个位置，而是把造成对 LUB 需要的对不同类型（例如类类型 X_1，X_2，…，X_n）的引用用符号列出，可能用^之类的标记或符号隔离。该符号表示这些类型必须共享具有一个 LUB 的这种关系。

约束实施

当和如果实际装载了被引用类 B 时，在步骤 650 实施，例如检验，VM 使用的写入或记录的约束。因此，在图 4A 中步骤 475 代表的后装载步骤期间装载之后立即实施约束。图 6C 示出了根据本发明的一个示例实施例的对被引用类 B 的验证约束的实施。本发明的这个实施例代表了一个可以包括图 4B 中所示的 475a 的各步骤的新处理过程 475b。实施在步骤 671 开始。在示例中，把 B 的实际超类型用于确定 B 是否满足以前写入的子类型验证约束。在步骤 673 进行这个检验。如果被引用类 B 不满足约束，那么在步骤 674 发出一个错误。用于这个错误的处理程序可以终止执行或允许执行继续。作为选择，可以终止执行而不发出一个错误。如果被引用类 B 确实满足了写入的约束，那么在步骤 679 结束后装载处理。

通过这样的改进，VM 可以实现带有验证的完全迟缓装载。采用完全迟缓装载的优点包括：

- 有关链接错误的程序的行为对于所有平台和实现都是相同的。
- 不需要预期类或方法可能链接的所有位置，和试图捕捉所有这些位置上的异常。

•用户可以用可靠和简单的方式确定类或其它模块的可用性。

本发明的一个优点是，程序员可以用可靠和简单方式在一个平台上测试类的可用性。例如，如果程序员希望使用一个新版次的模块并且仅在这些新模块可用时使用它们，那么迟缓链接使其变得容易。在这种情况下程序员产生的代码中的某些地方会是对新版本的新模块的引用的支路。如果平台的当前版本不支持在该支路中引用的模块，那么程序员将把程序设计为不执行该支路。如果新模块在该平台上不可用，并且验证的模块引用该新模块，那么不是完全迟缓的虚拟机可能需要验证程序尝试装载该遗漏的新模块。这个装载必然失败，它将导致验证步骤的失败。因此，由于一个遗漏模块，验证将造成失败，即使该模块仅在一个不会执行的支路中引用。利用需要的完全迟缓装载，由于并不实际执行的指令引用的模块，验证不会失败。在检验类这样的模块的最新版本的同时通过验证的能力提供了对采用本发明支持的完全迟缓装载的极大推动，这是虚拟机所需要的。

即使利用需要的迟缓装载，VM 的不同实现可以自由地提前装载和链接——只要任何失败仅在合法的定义点上出现。代码必须能够执行到带有错误类型的类必须决定的点。例如，一个适时（JIT）代码发生器可以选择在编译时预链接一个类或方法。但是，如果不是立即失败的任何链接失败，JIT 应当产生将使适当的异常在迟缓装载应当已经完成（它不须要实际进行链接时间检测，尽管它可以）的点发生的代码。作为另一个示例，由于无效类引用，一个静态编译程序在其专用链接阶段可能失败。尽管如此，如果即使它不能够完全地链接，它选择编译代码，那么在执行时该代码在 JIT 编译代码的相同的点失败。作为最后一个示例，当动态地装载了一个类时（例如，通过一个类装载程序），一个实现可以选择完全或部分地预链接它。但是，如果有任何失败，代码也必须能够执行到无效引用的点。

逐模块验证

在本发明的另一方面，即使装载了被引用模块也不进行被引用模块的验证。有许多原因希望使用逐模块验证，也叫作一次一模块（类）验证。其允许在运行时间之前进行验证，具有有益结果。可以减少或一同消除在时间和空间上的验证运行时间代价。JAVA™ 运行时间环境可以更小和更简单，因为它包含的实现验证的代码量可以减少。现有 JVM 规定或上述提议的完全迟缓装载并不自动提供这种如同逐类验证一样实现了对 JAVA™ 平台的扩充的一次一模块验证。在第一种情况下，如果需要，验证程序无可选择地自动装载被引用类。在后一种情况下，如果装载了被引用类，将发生被引用类的验证，并且这种验证也是无可选择地进行。因此预期有逐类验证的两种实施例，一种可以在现有 JVM 设计使用，另一种可以在新的完全迟缓装载设计使用。

根据本发明的一个实施例，可以在运行时间之前进行通常在验证期间进行的检验。因为运行时间之前的检验在技术上不是链接部分，因而不是链接的验证阶段部分，这些检验在这里被称为预验证，表示有效性的潜在预运行时间检验。

在本实施例中，在编译程序产生了一个二进制类之后的任何时间，程序员可以进行有关该二进制类的预验证，并且独立地进行对在该类中可能要引用的任何其它类的验证。结果，不需要访问被引用类或被引用模块。图 7A 中示出了本发明的逐类预验证。在步骤 710，例如，通过把类 BAR 从存储介质装载到存储器开始该方法。然后，在步骤 712 进行有效性检验，例如类型检验，操作数堆栈溢出/下溢检验，这些检验在现有技术中是在验证过程中进行的。在检验期间，乐观地假设引用其它模块的指令验证所需的任何模块间信息，诸如从类 BAR 引用的类 A 和 B 之间的子类型关系，从而使该指令是有效的。但是，假设的信息或关系把一个约束置于虚拟机必须记忆的被引用模块上。如果最终装载了这样一个被引用模块，那么必须检验这些约束。如果尽管有这些假设，类 BAR 之类的模块没有通过所进行的检验，

则产生一个错误，步骤 713。由于这样一个错误不必是一个运行时间错误，在执行期间可以不把它发出到一个处理程序。而是可能必须把它传送给程序员，例如利用出现在打印机或显示屏幕这样的输出设备上的错误消息通知给程序员。如果模块通过了所有检验，那么在步骤 716 写入任何要调用的预验证约束，以便以后在运行时间使用。接下来，当预验证完成时在步骤 719 停止处理过程。然后可以按照图 7A 中所示步骤预验证另外的类或模块。并不是模块中所有指令都需要预验证，只是模块的特殊使用所需的指令才需要预验证。例如，可能只需要验证方法 FOO 中的指令，但类 BAR 中其它方法中指令不需验证。

作为选择，可以把预验证约束写入一个文件，或用其它方法与模块联系，以便以后在运行时间检验。为了 JVM 使用，可以把这些约束作为与该类相关的数据以二进制类格式记录在存储介质上。当把类装载到存储器中时，JVM 可以把这些约束内在化，以便在需要时，例如在装载了被引用类 B 之后检验。

作为另一种选择，根据本发明，无论是存储的预验证约束或是模块本身或是二者都可以附加一个信号，例如数字签名，可以用于可靠地识别模块或约束的源，和指示自标记以来它们是否被改动。

以这种方式，可以在运行时间之前进行相当于现有验证期间进行的，但是不需要关于类 A 和 B 之类的被引用模块的跨模块信息的有效性模块内验证检验。也就是说，对于模块内检验可以进行实际上的完全逐模块预验证。模块间检验转变为预验证约束。

图 7B 示出了用作示例的图 7A 的步骤 716 的详细情况。在步骤 732，处理过程开始所装载的类 BAR 的一个方法的预验证。接下来，在步骤 755 确定该方法中的下一个指令是否需要来自被引用类 B 的信息，以便使该指令能够进行其有效性检验。如果不是，那么在步骤 737，程序进行有关该指令的任何需要的类内有效性检验。如果指令未通过

类内检验，那么把一个错误消息写入输出设备。然后程序员可以处理这个问题。另一方面，如果必须装载一个被引用类以充分验证该指令，那么在步骤 750 写入或用其它方式记录一个预验证约束，以便以后调用，并且假设该指令所需的子类型关系有效。由于该指令可能也需要类内检验，控制前进到步骤 737，进行这些检验。如果该指令不需要类内检验，那么在步骤 737 它自动地“通过”检验。如果在类内检验后发现该指令有效，和/或在写入预验证约束后假设其有效，那么流程控制转移到步骤 782，步骤 782 循环到步骤 755，直到所装载的类 BAR 的方法 FOO 中不再有剩余指令，此时方法 FOO 的预验证结束。应当注意，无论是进行模块内检验还是写入预验证约束，都不进行是否装载了被引用类的确定。即使是对于已经装载的模块也不进行跨模块检验。

图 7C 示出了如何在运行时间通过链接期间进行的验证来处理已经在运行时间之前一次一模块地验证过的模块，例如类 BAR。为了取代现有 JVM 的步骤 530a，或用于完全迟缓装载的修改的步骤 530b，图 7C 示出了一个遵循现有 JVM 的近乎迟缓装载并且结合逐类预验证的替代步骤 530c。在步骤 792，在对类 BAR 的方法 FOO 之类的模块的指令的验证步骤开始之后，在步骤 780 确定该模块是否已经通过预验证。如果没有，那么控制跟随在图 5B 中的步骤 555 开始的现有 JVM 的流程。否则，在步骤 783 可选地检验预验证的模块是否是可信之后，下面将对其更详细地说明，控制前进到步骤 784。已知有许多测试一个文件是否可信的现有技术方法，例如通过使用数字签名。如果不关心预验证模块的可信性，那么可以省略步骤 783。在步骤 784，并不是一步步地通过方法中的指令，而是运行时间验证程序读出在 BAR 的逐类验证期间记录/写入的预验证约束。如果没有写入的预验证约束，那么 BAR.FOO 的验证完成，控制前进到步骤 778，完成处理过程。

如果为一个被引用类，例如，类 A 或 B，写入了一个预验证约束，

那么在步骤 786，运行时间验证程序确定约束中的被引用模块是否已经装载。如果是，那么在步骤 788 实施该预验证约束。如果被引用模块未通过该约束，那么在步骤 762，发出一个错误使一个错误处理程序能够捕捉到该错误。否则控制前进到步骤 778，该步骤循环通过预验证约束，直到没有剩余的约束。如果在步骤 786 确定没有装载被引用模块，例如被引用类，那么在步骤 789 装载该被引用模块（例如一个类），并且在步骤 788 实施约束。

只要已经预验证了类（并且作为选择，通过了可信检验），无论是否写入了有关被引用类的预验证约束，都不需要进行类内检验；它们已经在运行时间之前的逐类预验证期间进行了检验。然后，根据本发明，在一个模块通过了一次一模块的预验证之后，运行时间验证不进行模块内检验；它仅实施模块间约束。

在所述的示例中，在模块编译后立即进行预验证，而不装载任何其它模块。这使得在运行时间之前能够进行大量验证，并且不用在每次模块装载和链接时重复，因而节约了宝贵的时间和空间资源（例如，在运行虚拟机的处理器上）。

<u>带有完全迟缓装载的逐模块预验证</u>

图 8 描述了结合在运行时间完全迟缓装载期间预验证结果的流程图。图 8 示出了对于示例类 BAR，在支持完全迟缓装载的链接期间进行的验证是如何处理已经在运行时间之前一次一模块地验证过的模块的。替代现有 JVM 的步骤 530a，或用于完全迟缓装载的修改的步骤 530b，或带有一次一模块验证的近乎迟缓装载的步骤 530c，图 8 示出了遵循 JVM 的一个新实施例的完全迟缓装载并且结合逐类预验证的替代步骤 530d。图 8 中的步骤 892，880，883，884，878，886 和 862 与图 7C 中的对应的 700 号的步骤 792，780，783，784，778，786 和 762 分别类似。在步骤 892 开始对一个如类 BAR 的方法 FOO 这样的模块的指令的验证步骤之后，在步骤 880 确定该模块是否已经通过预

验证。如果没有，控制遵循图 6A 中的 JVM 在步骤 655 开始的完全迟缓装载的流程。在步骤 833 可选地检验预验证的模块是否可信之后，如前面说明的，控制进入步骤 884。在步骤 884，并不是逐步通过方法中的指令，而是由运行时间验证程序读出在 BAR 的逐类验证期间写入的预验证约束。如果没有写入的预验证约束，BAR.FOO 的验证完成，并且控制前进到步骤 878，完成处理过程。

如果读出了用于一个被引用模块，例如类 A 或 B 的一个预验证约束，那么在步骤 886 运行时间验证程序确定是否已经装载了约束中的被引用模块。如果是，那么在步骤 888 实施预验证约束。如果被引用模块未通过该约束，那么在步骤 862 发出一个错误，由错误处理程序捕捉。如果被引用模块无需验证而通过约束，那么流程前进到步骤 878，步骤 878 循环通过预验证约束直到没有任何剩余。

图 8 中剩余的用于完全迟缓装载的步骤实际上与用于近乎迟缓装载的它们的对应部分不同。如果在步骤 886 确定没有装载被引用模块，例如被引用类，那么不装载该被引用模块。而是在步骤 889 把预验证约束复制到，或用其它方法保存到存储器或存储介质中，以便在装载现在尚未装载的模块（例如一个类）时，实施该约束。

在图 8 中，步骤 888 的实施可以有三种结果。除了失败和无条件通过之外，可能有已经装载的被引用模块只有在知道一个或更多的尚未装载的模块的内容时才能通过的结果。这种结果可以看成是“有条件通过”，有条件通过是把有关一些被引用模块的预验证约束重写为有关尚未装载的（一个或多个）被引用模块的验证约束。步骤 885 把预验证约束重写为仅有关于尚未装载的被引用模块，例如类，的验证约束。重写之后，如果需要，控制前进到步骤 878。

<u>非置信类的逐模块验证</u>

如上所述，根据本发明的验证依赖于能够用被引用模块所必须满

足的约束来构造和注释一个模块的能力。不幸的是，程序并不总能防止攻击者对这种注释进行电子欺骗——可能使一个恶意类看起来是良性的。因此，根据本发明的一个实施例，可选的可信检验包括在图 7C 的步骤 783 和图 8 的步骤 883 中。缺少这些检验，在可信场合可以使用预验证，例如在执行之前可以对类进行预验证并将其装载到非置信（防篡改）数据库中的场合。

但是，在非置信场合，需要更多的防护。根据本发明的一个实施例，如图 9 中所示，建立一个高速缓冲存储器。高速缓冲存储器 920 包含诸如可信的类和/或预验证约束之类的可信模块，例如 965a。从一个非置信源，例如因特网上的源，输入到虚拟机的模块和/或约束将放置在高速缓冲存储器外部，例如在 965。从例如 965 的非置信源与类一同输入的任何预验证约束将被忽略。而是在第一次装载这样一个模块时，根据图 7A 的所示方法，在一预验证程序 910 中急切地预验证它。如果模块未能通过预验证，它将被立即拒绝。如果模块通过预验证，则按需要产生新的预验证约束，并且随后把这个注释的或与新约束关联的模块，例如 965a，存储在一个可信模块高速缓冲存储器 920 中。在以后试图从一非置信源装载模块时，将首先搜索模块高速缓冲存储器 920。如果发现了缓存的预验证模块 965a，那么可以把模块 965a 安全地用作预验证模块。利用这种改进，图 7B 中所示的逐类预验证约束的检验将会正确地进行。实际上，图 7C 的步骤 780 回答了关于是否已经通过检验模块高速缓冲存储器进行了预验证的问题。利用这种改进，不需要图 7A 中的步骤 718 中的预验证约束的数字签名。同样，利用这一改进，图 7C 中的步骤 783 和图 8 中的步骤 883 中所示的预验证输出是否可信的检验也不需要了，并且流程直接分别从步骤 780 或 880 前进到步骤 784 或 884。

约束的形式

图 6A，6C，7A，7B 和 8 中的流程说明的方法都提供了尽可能迟地检验被引用类的要素。写入的约束的形式，以及随后检验这些约

束的方式如下。可以在例如，图 6A 的步骤 650，图 7B 的步骤 750，和图 8 的步骤 885 及 889 中写入约束。约束的实施可以在图 6C 的步骤 673，图 7C 的步骤 788，和图 8 的步骤 888 中应用。

5 下面通过举例更详细地说明约束产生和约束检验。参考图 2，赋值语句表明将把类 B 的一个新实例存储在类类型 A 的变量 var 中。在一种面向对象的语言中，这种赋值需要 B 是类 A 的一个子类型，如表达式 B<=A 代表的。这在 BAR 验证期间是未知的，除非在那时装载了 B。如果装载了 B 并且 B 是 A 的子类，那么 A 也必须装载（因为
10 要装载 B 必须装载 A）。因此，倒置是真，也就是说，如果装载了 B 和未装载 A，那么 B 不是 A 的一个子类；赋值语句引起了造成类 BAR 验证失败的子类型失配。如果 A 和 B 都装载了，如图 3 中所示的急切装载，那么可以跟踪 B 的超类的指示符，以查看 A 是否在某处是 B 的一个超类。如果是，那么 B 是 A 的子类，并且这个赋值通过验证。
15 如果通过跟随超类指示符在分层结构中向上搜寻未发现 A，那么 B 不是 A 的一个子类，赋值语句造成子类型失配，并且类 BAR 验证失败。

利用现有 JVM 规范，如果类 B 尚未装载，验证程序将装载类 B 并检验它的类型，具体讲，检验它是否是类 A 的子类型。

20

根据本发明，为了取得完全迟缓装载，或逐类验证，或二者，希望不装载类 B。因此，根据本发明的这个实施例，不装载 B，而是写入约束 B<=A。可以在上面列出的用于写入约束的任何步骤中（例如，650，750，889，885）写入该约束。以后在执行 BAR.FOO 时，如果
25 不执行这个带有赋值语句的支路，并且 B 同样也不可能被其它执行的指令引用，那么不会装载类 B。但是，如果执行包括这个赋值语句的支路，并且 B 尚未装载，那么此时装载类 B，并且在此时装载了类 B 之后，进行类 B 是否满足约束 B<=A 的检验。这种检验可以在，例如，上面列出的用于检验约束的步骤中（例如，673，788，888）进行。
30 这将很容易进行，因为如果类 B 确实是类 A 的一个子类，并且从类 A

继承了它的属性，那么类 A 必须是已经装载的。因此，这种类型的约束允许完全迟缓装载，逐类预验证，或二者。

根据本发明，存在有关非局部类的类类型的另一种检验，其可以在逐类预验证中以不同于完全迟缓装载实现中的处理方式处理。这是对受保护成员的接收方访问检验：

在 JAVA™ 虚拟机中，如果并且仅在下列条件下一个类或接口 D 才能访问受保护成员 R：

1.D 与声明 R 的类 C 在同一个运行时间封装中，OR BOTH

2.D 是声明 R 的类 C 的子类，AND

3.如果 R 是一个实例成员，那么 T，被访问的实例 R 的静态类型，是 D 的子类型。

要求 3 称为接收方受保护检验。

在现有 JAVA™ 虚拟机中，在 D 的链接期间当决定从 D 到 R 的引用时检验头两个要求，而第三个要求是在 D 的验证期间检验的。在验证期间，声明 R 的类 C 可能尚未装载。在这种情况下，显然 C 不是 D 的超类（否则，C 必然已经装载，因为装载一个类意味着装载了它的所有超类）。在这种情况下，只有 C 和 D 在同一运行时间封装中时访问才是合法的。验证程序可以乐观地假设这是成立的。当决定引用时要检验上述要求。因此，如果 C 已经装载，验证程序仅需要进行受保护的接收方检验。在这种场合，完全可以确定 R 是否是一个受保护成员。如果 R 不受保护，则不需要受保护的接收方检验。如果 R 是受保护的，验证程序可以测试，以确定 D 是否与 C 在相同的运行时间封装中。如果是这种情况，那么访问是合法的，并且也不需要受保护的接收方检验。如果 D 和 C 不在同一运行时间封装中，那么验证程序检验 D 是否是 C 的子类，和 T，被访问的实例的静态类型，是否是 D 的子类型。如果不是，则产生一个错误。应当注意，检验 T<=D 可能需要装载 T，如果它尚未装载的话。

在带有完全迟缓装载的验证中，当验证 D 时，假设已经装载了它的超类。除了一个例外之外，控制以与非迟缓情况相同的方式进行。如果确定需要一个是否 T<=D 的检验并且 T 未装载，那么必须避免装载。而是给 T 施加装载约束 T<=D。

在逐类验证中，情况不同。既未装载 D 的超类，也未装载声明 R 的类 C。因此，不进行受保护的接收方检验。不能作出如果 C 是 D 的一个超类则它必须已经装载的假设，因此不能检查 R 的声明。其结果是，甚至不能确定 R 是否是受保护的。作为替代，必须产生将在以后执行程序时检验的适当的约束。这一问题可以通过产生条件约束来解决：

If(D<=X)then{if(X.m protected)then{T<=D}else{true}}else{true}

用于每个以下形式的指令：

invoke o，X.m，

其中 o 具有类型 T。把类似的策略应用于域引用。在 D 的初始化之前检查这个约束。在该点，可以决定 D<=X（由于 D 和它的所有超类已经装载）。如果 D<=X 不是真，那么不需要进一步的行动。如果 D 不是 X 的一个子类，那么 D 不可能是声明 m 的类 C 的子类。其原因是 C 必须是 X 的超类。其结果是，只有在 m 是不受保护的或 C 与 D 在同一运行时间封装中时对 X.m 的引用才是合法的。这将在决定了对 X.m 的引用时检验。如果 D<=X 是真，那么可以检验 X.m 是否是受保护的。如果 X.m 不是受保护的，那么不进行受保护的接收方检验。否则，可以进行是否 T<=D 的测试，如上所述，这将造成 T 被装载。

当把完全迟缓验证与逐类验证组合在一起时，遵循逐类验证过程，除了在评价以下条件约束时之外：

If(D<=X)then{if(X.m protected)then{T<=D}else{true}}else{true}

如果必须评价 T<=D 和如果 T 未装载，那么应当如在迟缓场合中那样，把装载约束 T<=D 施加于 T。

当验证在一语句检查操作数堆栈的状态时，可以使用另外的约束，所述语句是几个以前执行的语句的后继语句，也就是说，在两个或多个支路的汇聚处。在这点，验证当前被设计为合并操作数堆栈和来自当前指令是其后继指令的前驱指令的局部变量的快照。如果要引用在尚未装载的类中定义的类型，这总是逐类预验证中的情况并且在一些时候是完全迟缓装载中的情况，则类型点阵不可用并且不知道 LUB。遵循上述对图 6B 的步骤 638 的 LUB 的符号表示，可以用如下用符号表示的一个列表中的一个约束来代替“the LUB<=class T”这样的约束：

X_1^X_2^…X_n<=T

这可以通过提取公因子而成为各个类 X_i 的一系列约束：

X_1<=T，X_2<=T，…X_n<=T。

当执行装载的类的当前方法并且通过需要例如类 X_2 的决定的支路时，装载类 X_2，并且可以在此时检验约束 X_2<=T。作为选择，如果在装载 X_2 时 X_2 通过检验，那么可以重写列表上的一个约束，从列表删除 X_2。

如上所述，在几个步骤（例如，650，750，889，885）中任何一个期间写入约束，并且随后在几个检验步骤（例如，673，788，888）中的任何一个中检验约束。

利用 LUB 的这种符号表示，实际计算可能要用较长的时间到达收敛。但是，保证了处理过程收敛，因为 JVM 的一个类文件的常数池是有限的。因此，一个方法仅能直接或间接地引用有限数量的类型。结果，一个 LUB 的符号表示必然是类名的有限序列，X_1^…^X_n。这

又表示对于 JVM 来说通过类型推理算法的重复次数是有限的，因为重复将一直继续直到没有新类型可以加入到 LUB 中。

尽管已经对本发明进行了详细的说明和图示，但是应当清楚地知
5 道这只是图示和举例的方式，而不能认为是限制的方式，本发明的精神和范围仅受所附权利要求的限制。

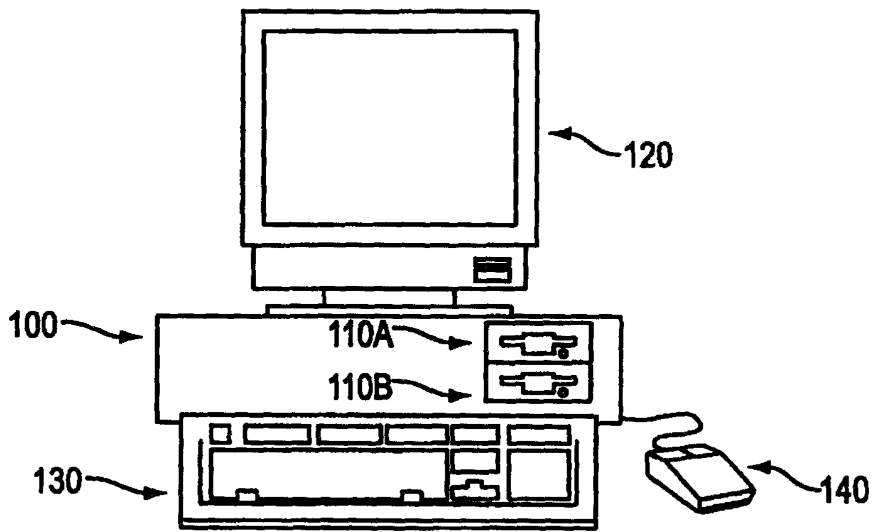


图1A

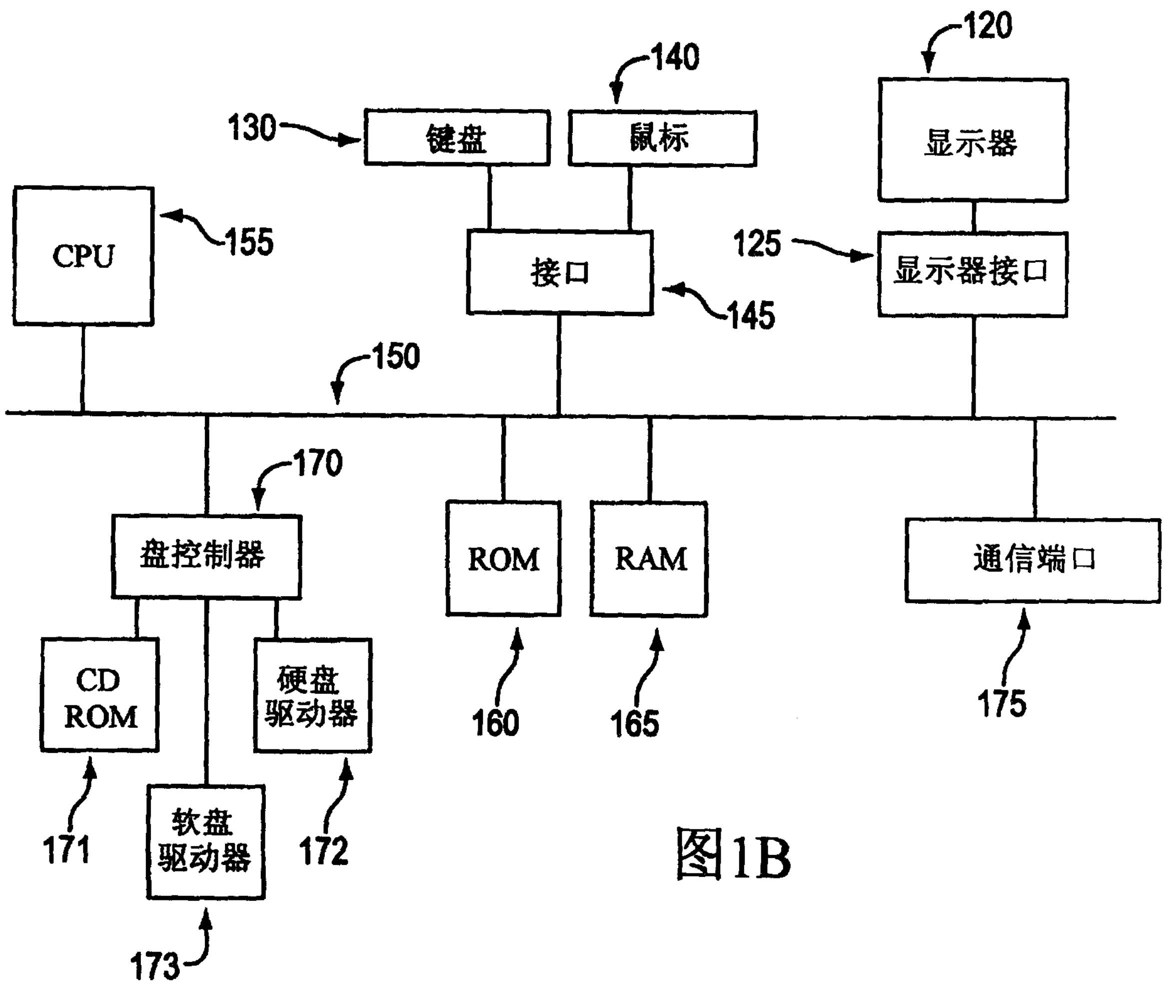


图1B

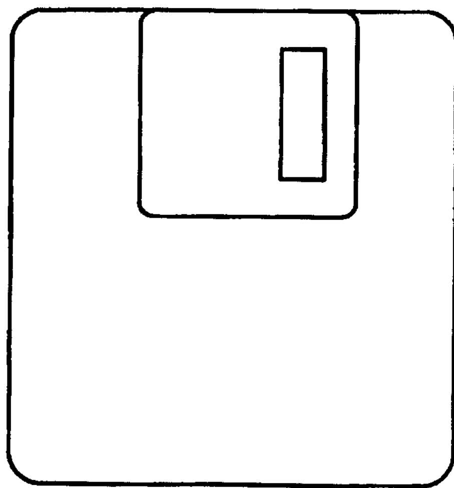


图1C

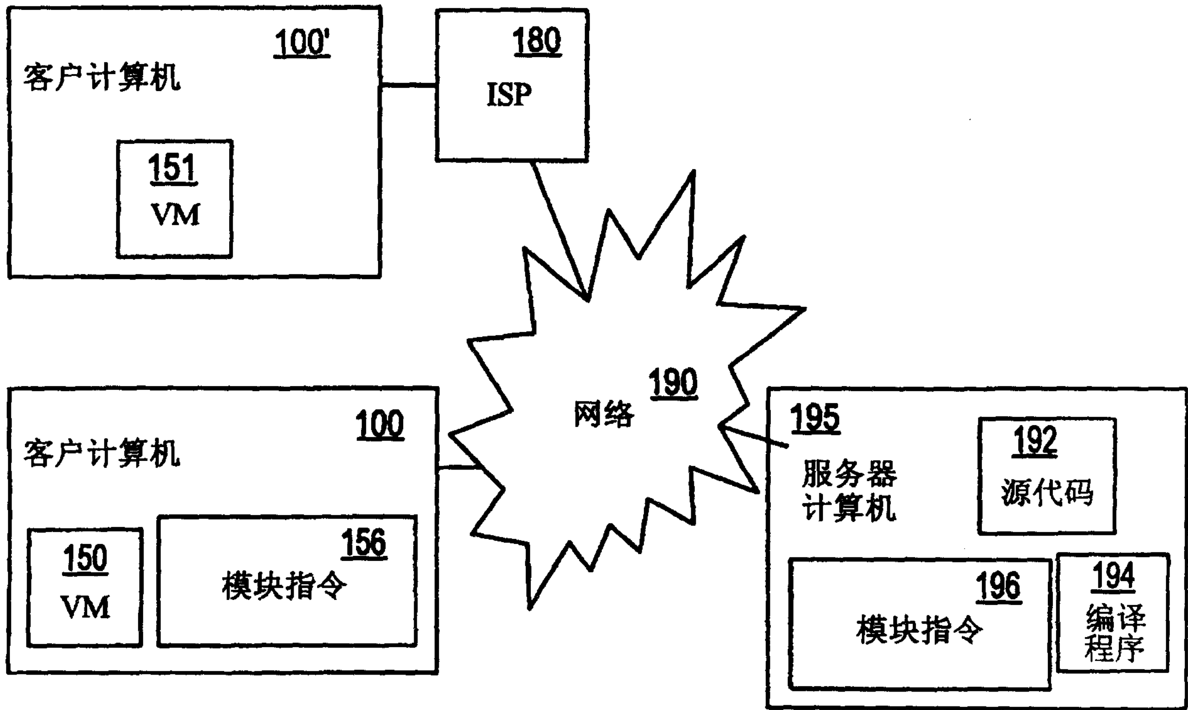


图1D

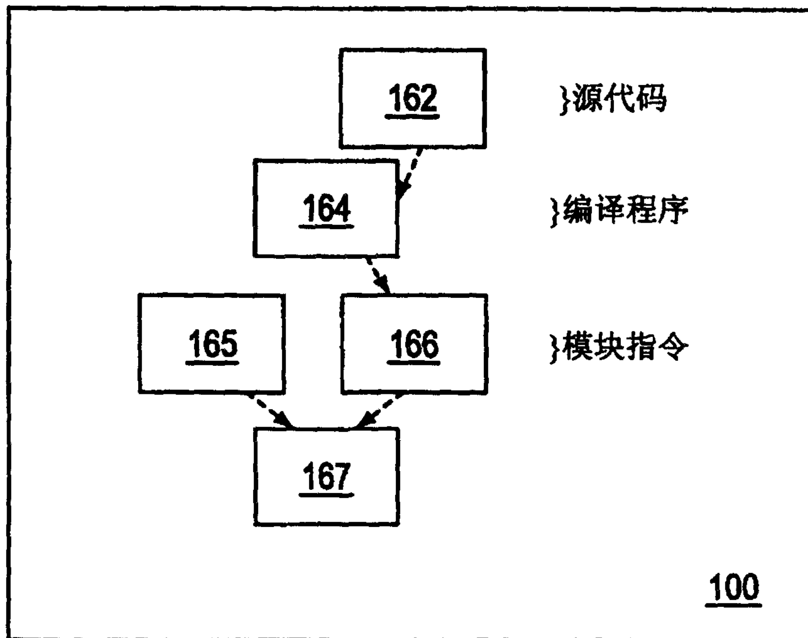


图1E

```
class BAR
  . . .
  void FOO(arg) {
    . . .
    if (arg){
       . . .
       A var = new B()
       . . .
    }
    else {
       . . .
    }
    z = z * z
  }
```

图2

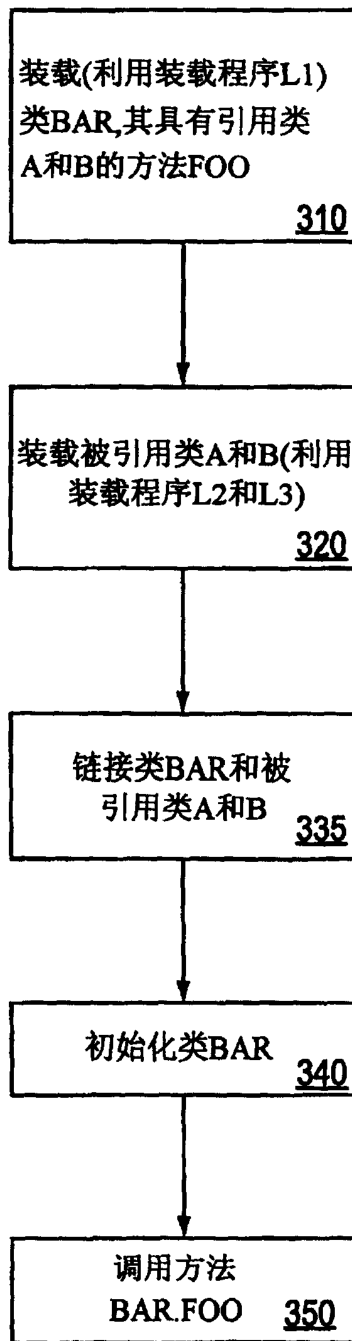


图3

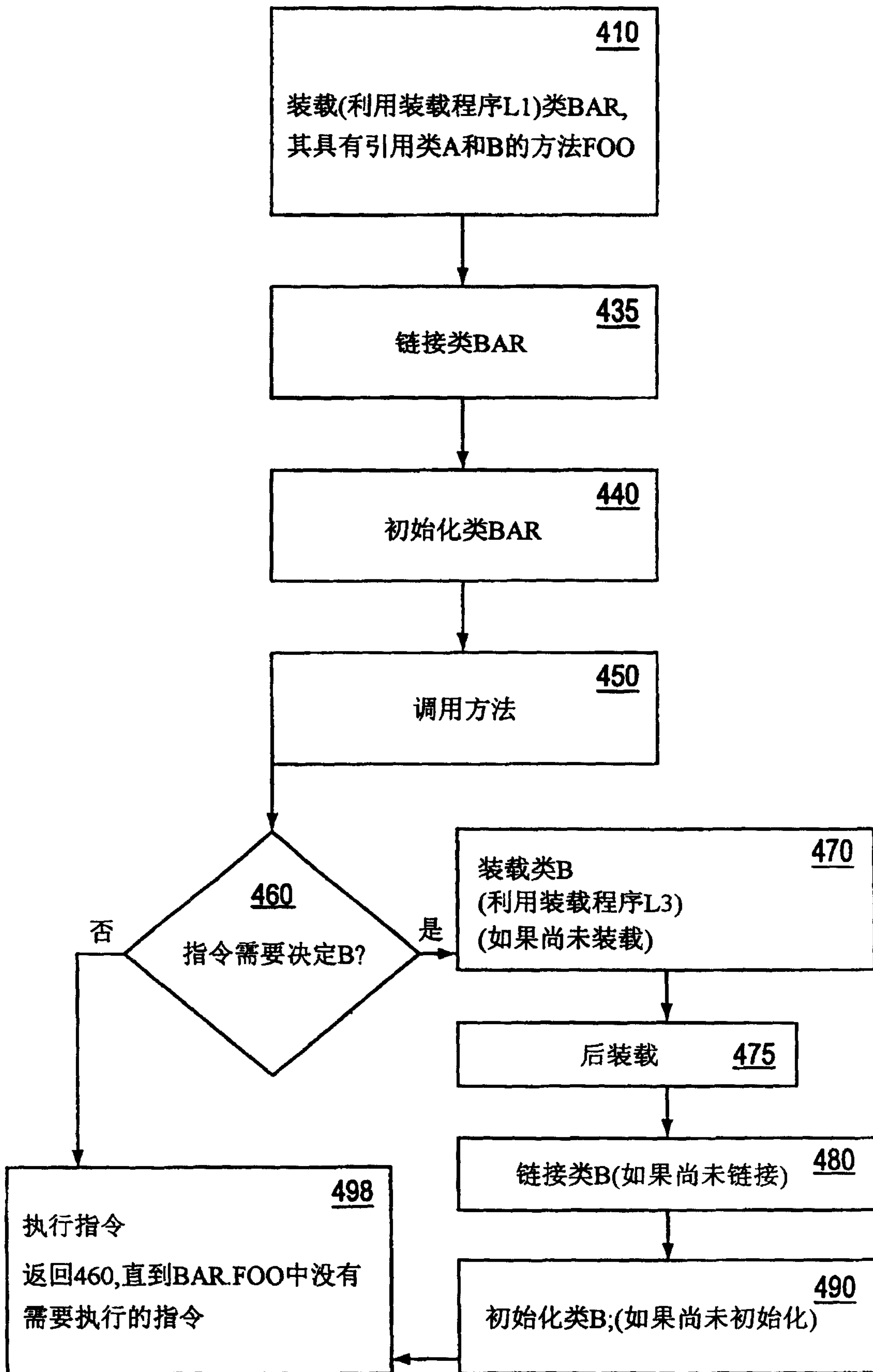


图4A

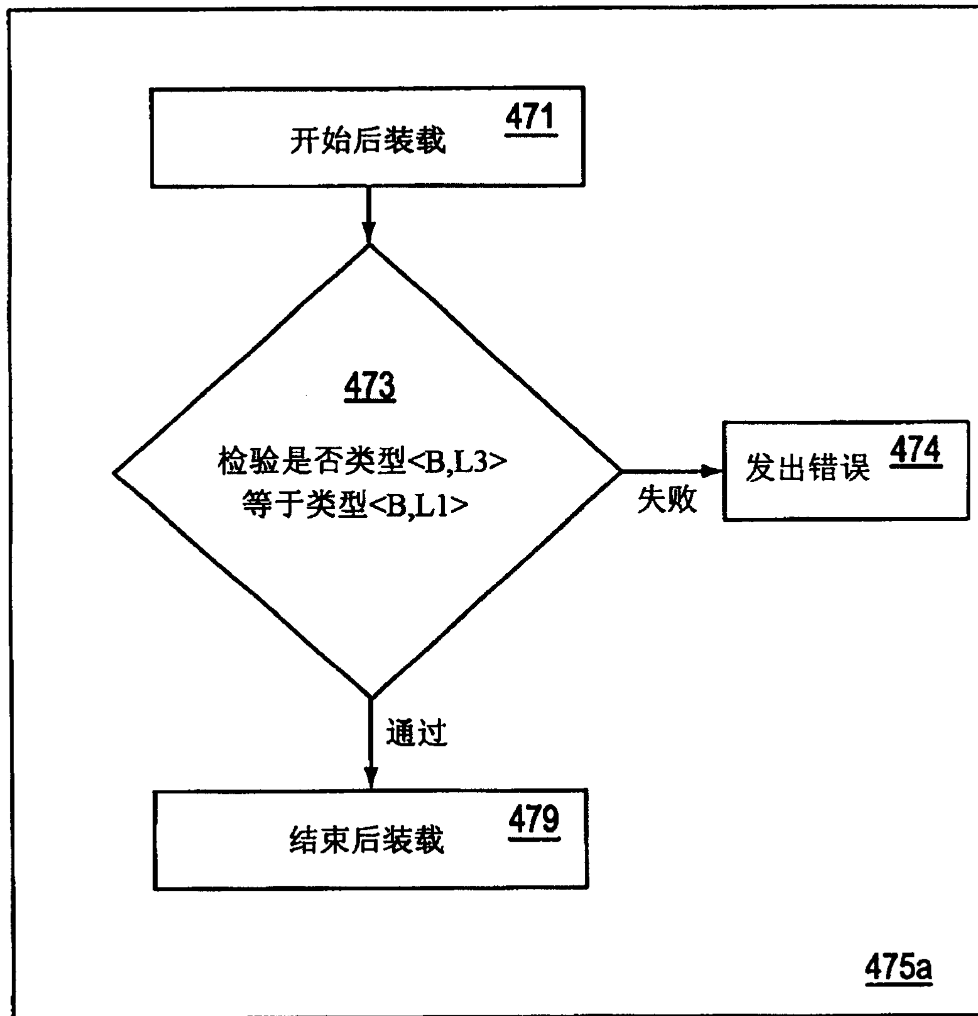


图4B

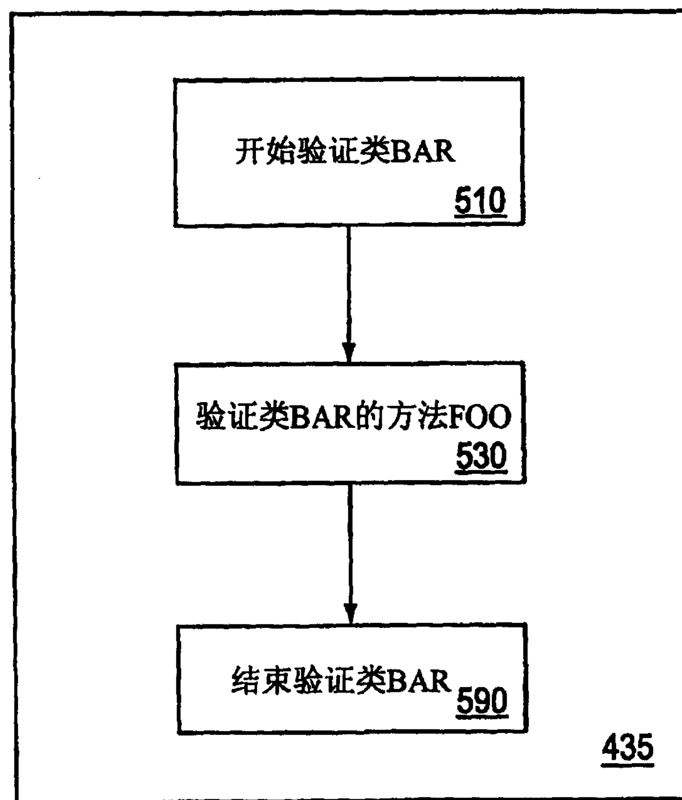


图5A

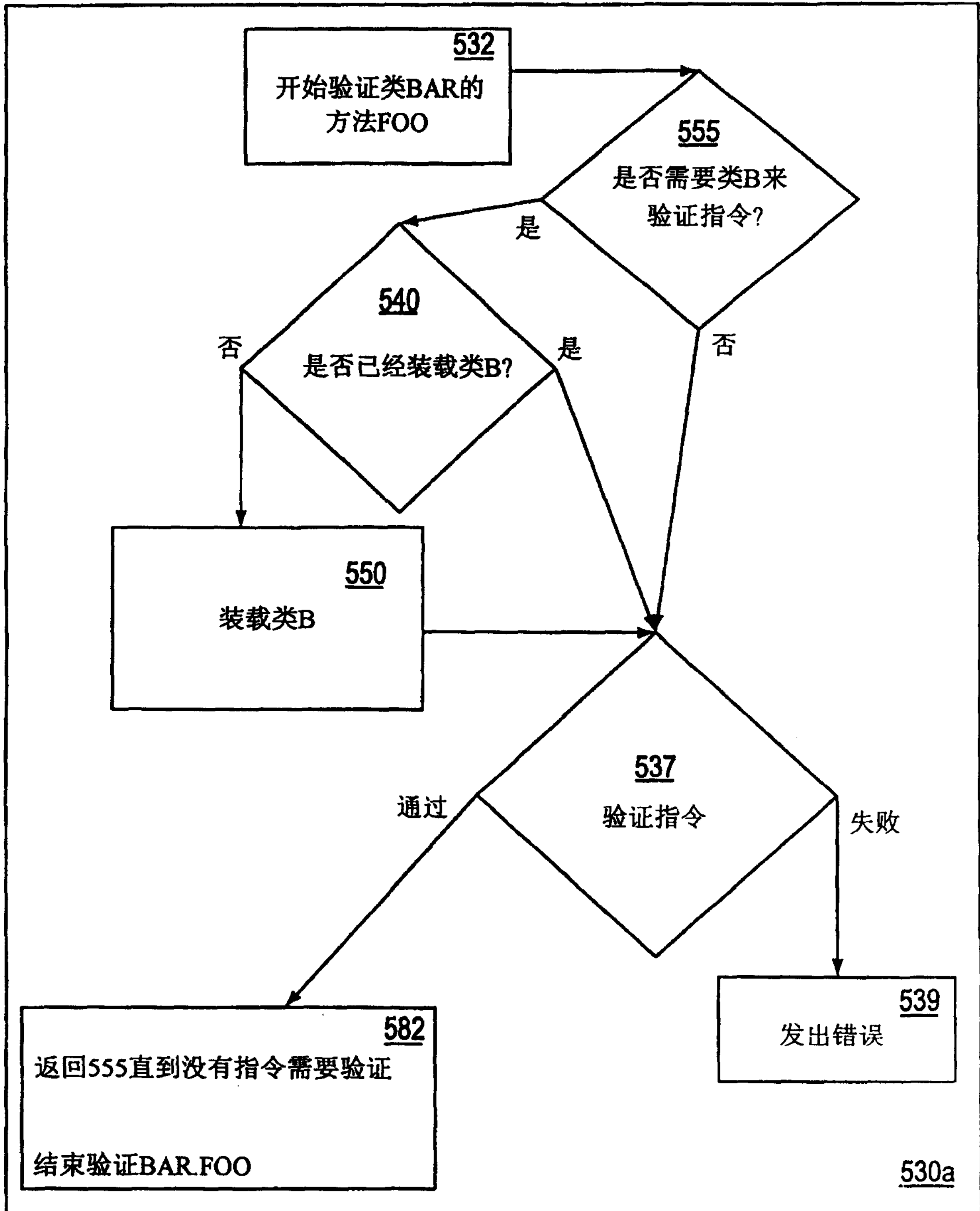


图5B

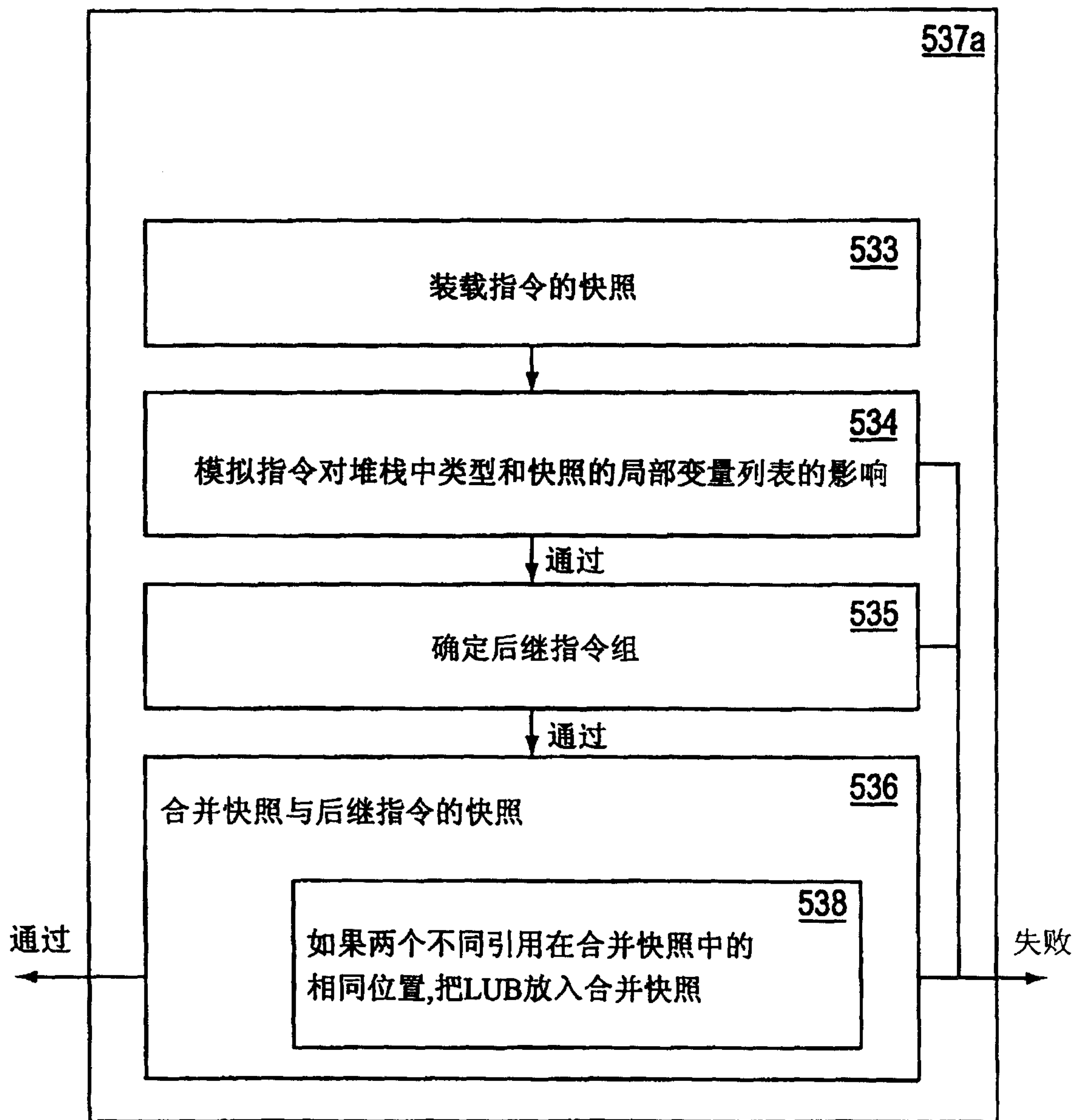


图5C

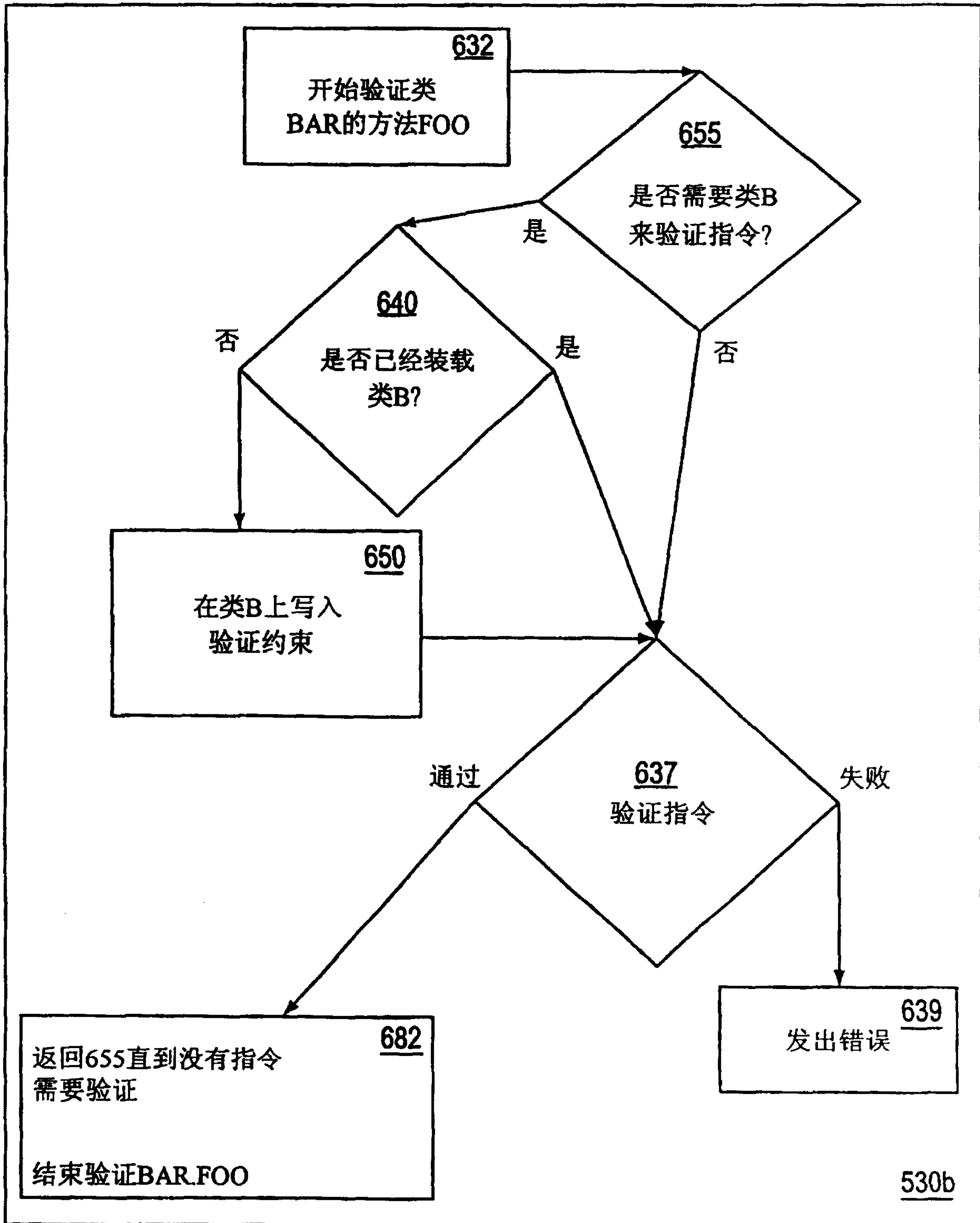


图6A

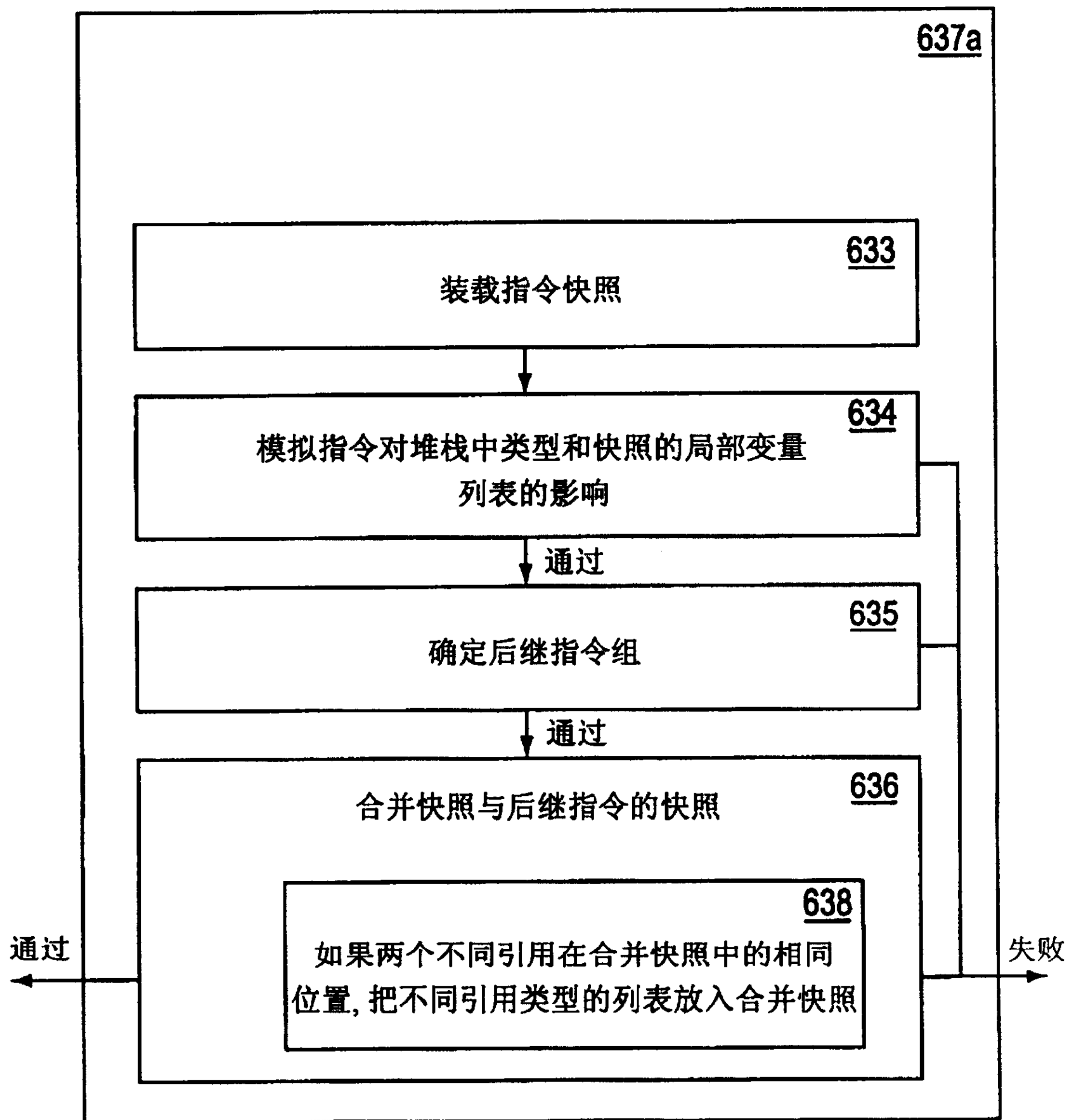


图6B

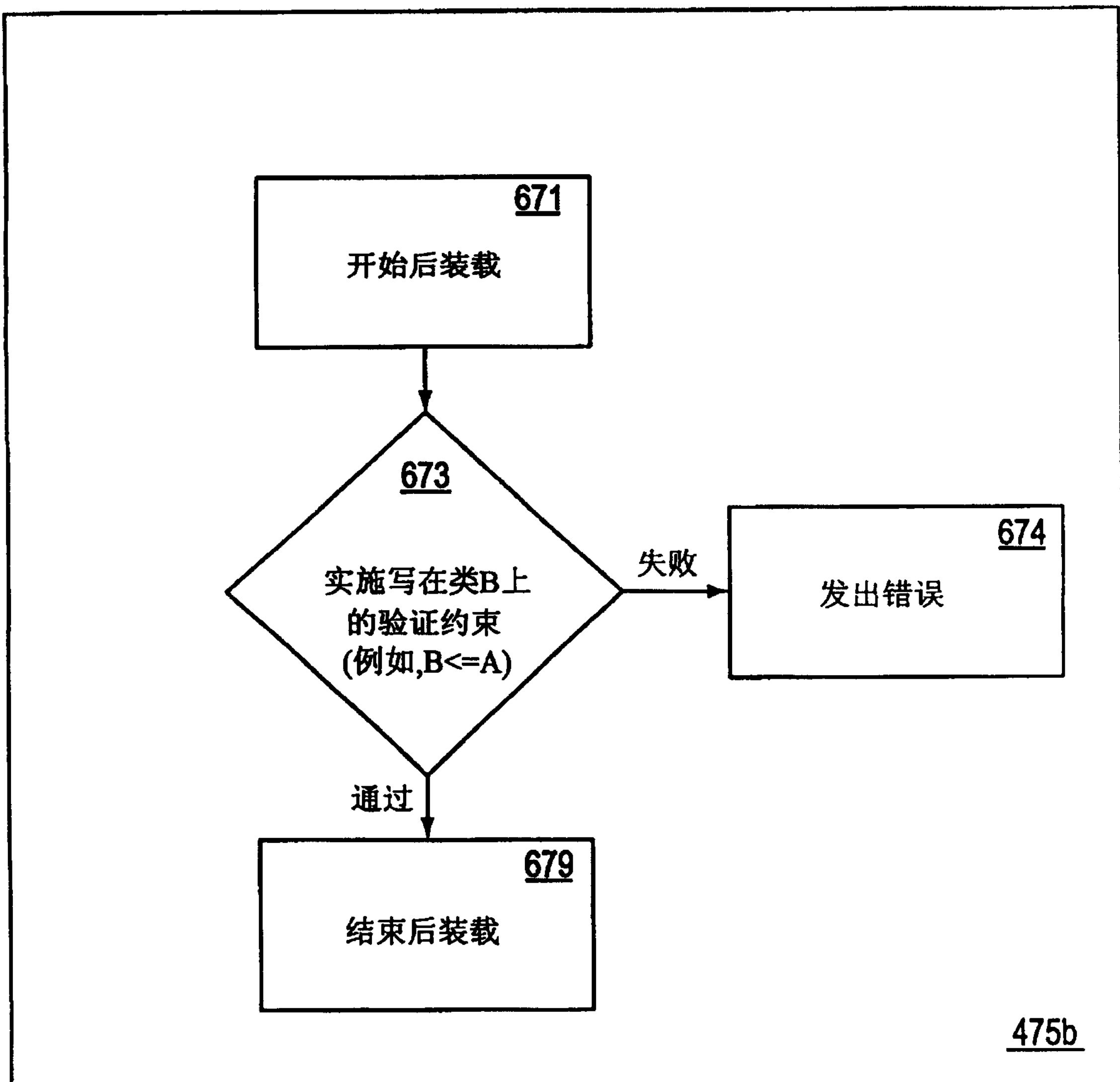


图6C

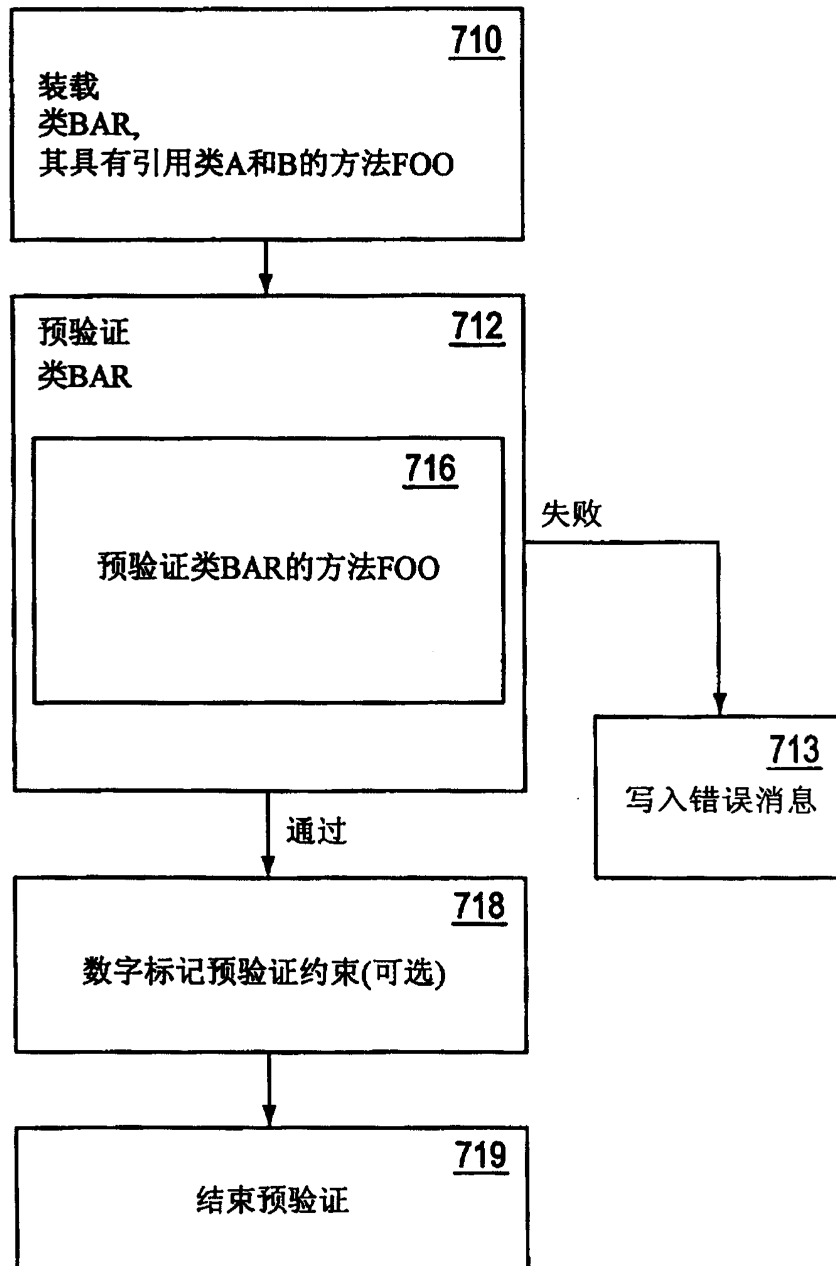


图7A

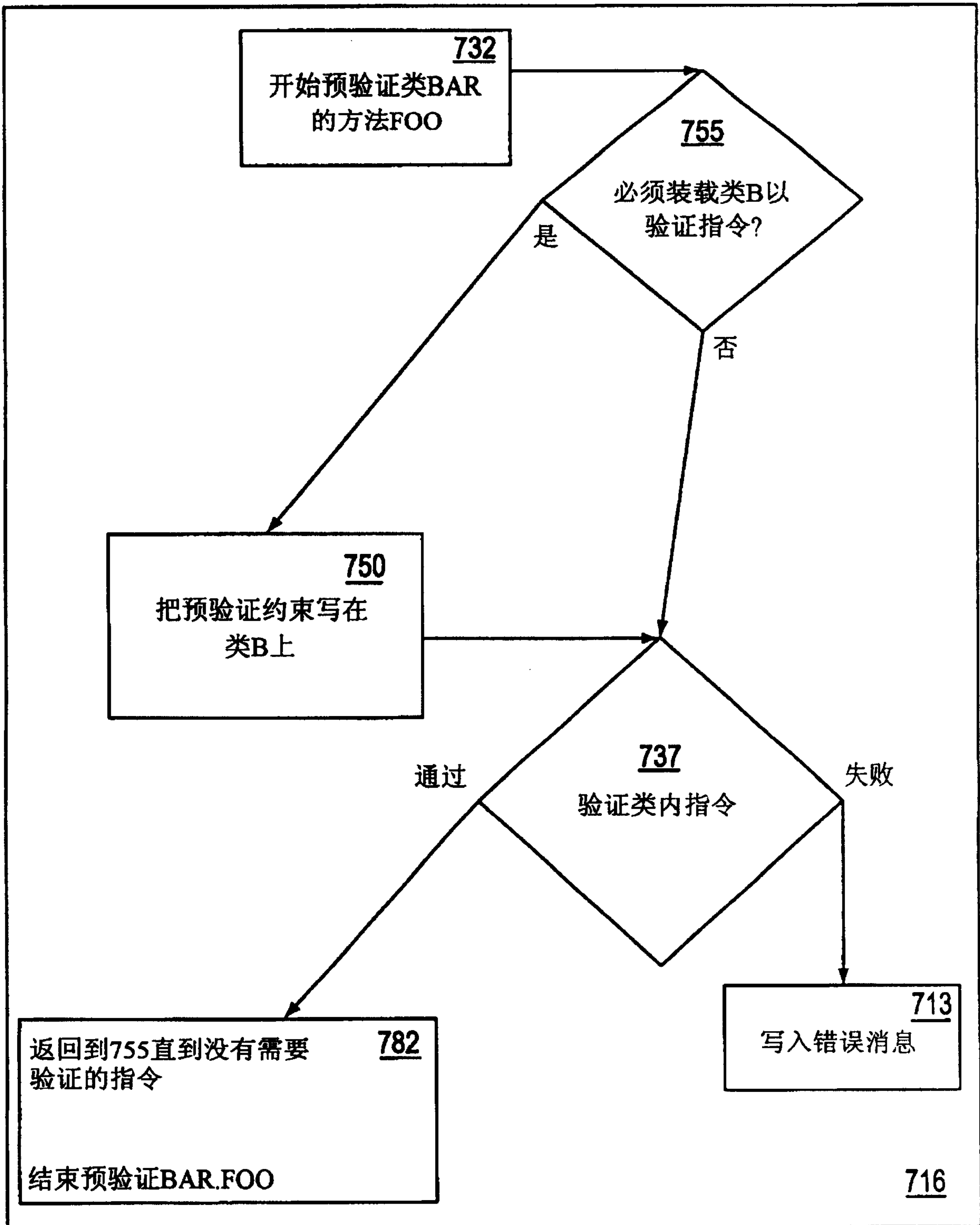


图7B

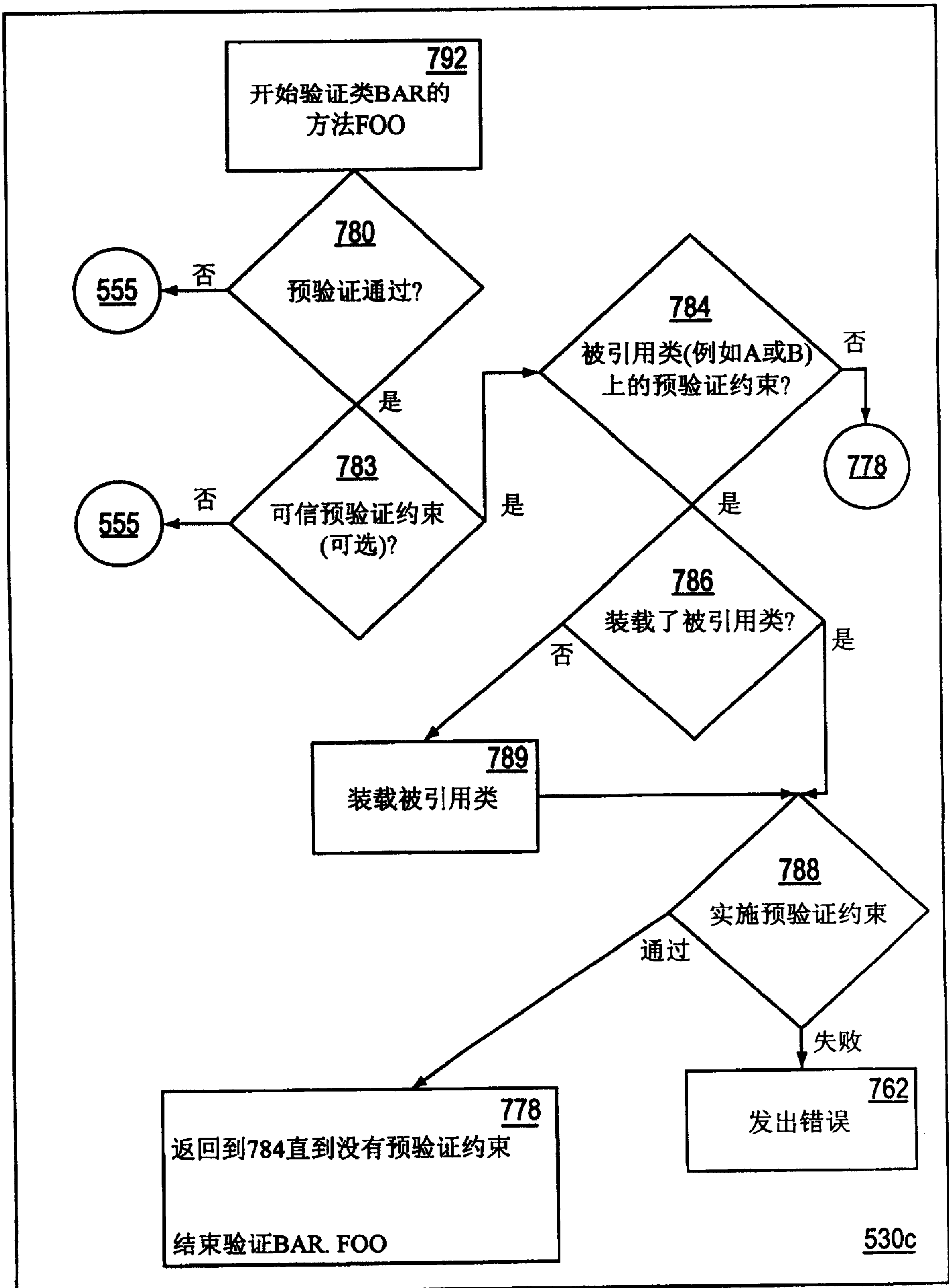


图7C

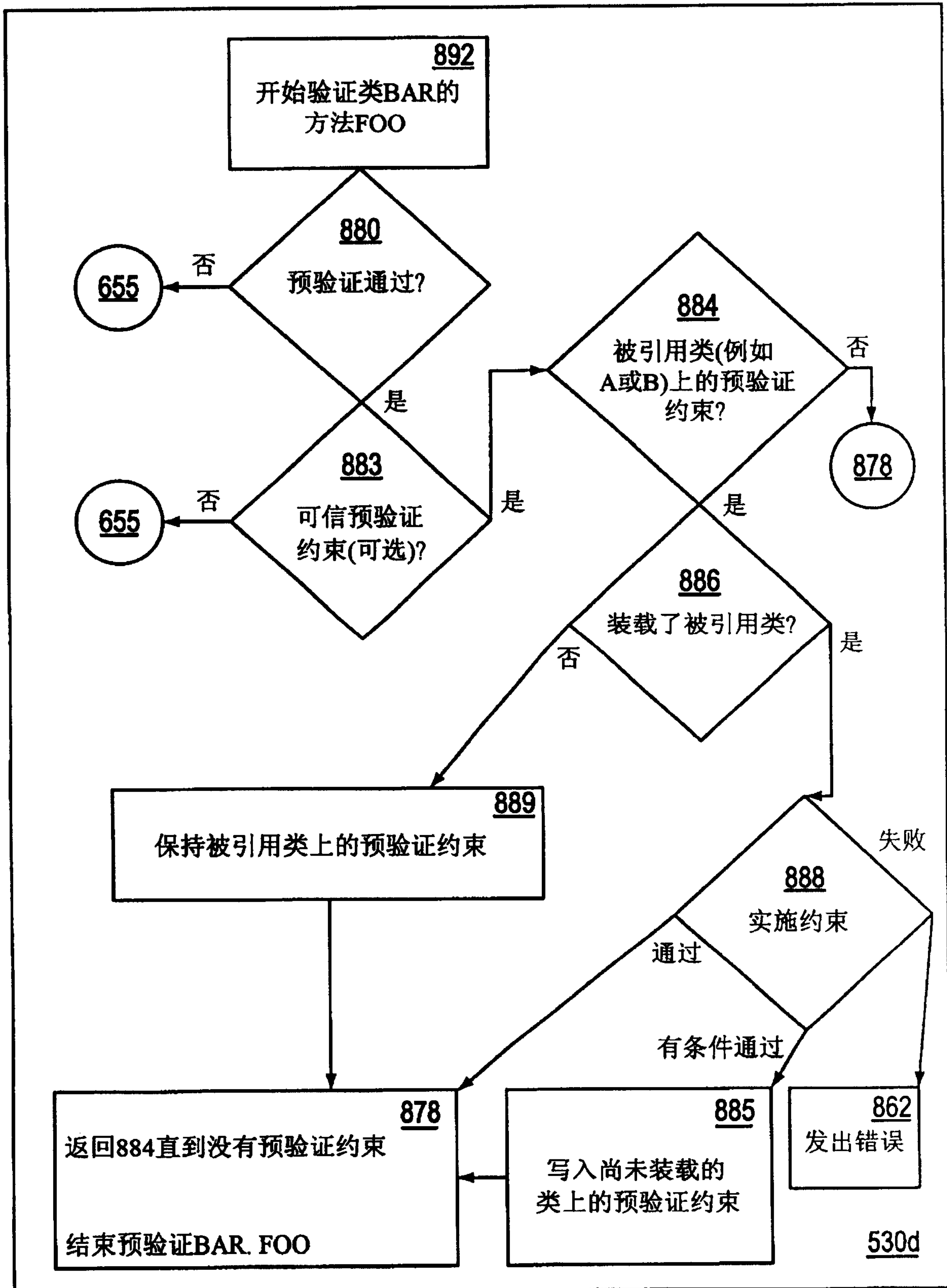


图8

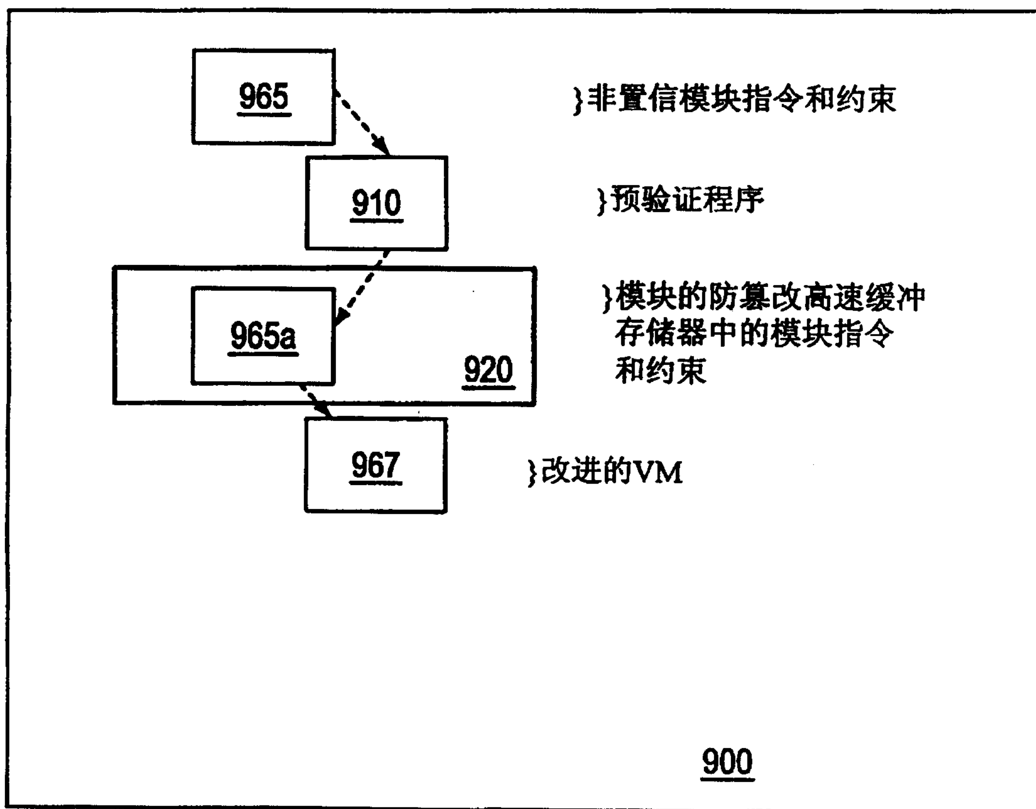


图9