[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/45 (2006.01)

[12] 发 明 专 利 说 明 书

专利号 ZL 200480018112.4

[45] 授权公告日 2009 年 4 月 1 日

[11] 授权公告号 CN 100474252C

[22] 申请日 2004.5.21
[21] 申请号 200480018112.4
[30] 优先权
[32] 2003.6.27 [33] US [31] 10/607,601
[86] 国际申请 PCT/US2004/015964 2004.5.21
[87] 国际公布 WO2005/006119 英 2005.1.20
[85] 进入国家阶段日期 2005.12.27
[73] 专利权人 微软公司
地址 美国华盛顿州
[72] 发明人 M·R·普莱斯科
小 D·R·塔迪提
[56] 参考文献
WO01/23996A1 2001.4.5
CN1305609A 2001.7.25
US5488727A 1996.1.30
US6381736B1 2002.4.30
审查员 魏军伟
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 张政权

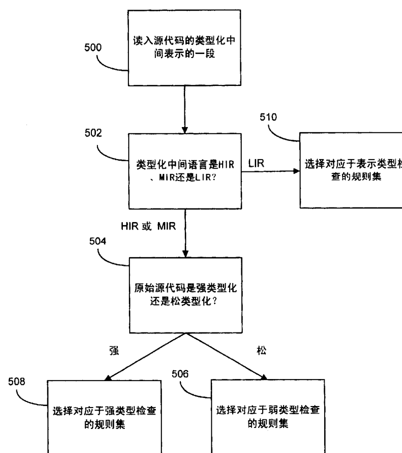
权利要求书 2 页 说明书 23 页 附图 8 页

[54] 发明名称

在编译过程中表示和检查程序组件的一致性的可扩展类型系统

[57] 摘要

提供了用于各种形式的中间语言校验的一致性的类型表示、类型检查器和编译器。在编译器中对编程语言进行类型检查是通过取一个或多个规则集作为对类型检查器的输入来实现的，类型检查器基于多个准则中的任一个或两个或多个的组合来选择一个或多个规则集。它们之中有编译阶段、源语言、体系结构以及进行类型检查的语言中存在的类型化级别。语言然后使用所选择的一个或多个规则集来进行类型检查。规则集可包括对应于强类型检查的一个规则集、对应于弱类型检查的一个规则集、以及对应于表示类型检查的一个规则集。作为替换，可提供具有基于先前提到的准则的的任一个，或两个或多个的组合从一较大的规则集在运行时选择一个或多个规则集的类型检查器的编译器。



1．一种依照一个或多个类型检查规则集在编译器中对编程语言进行类型检查的方法，包括：

基于当前编译阶段选择一个或多个类型检查规则集；以及

基于所选择的一个或多个类型检查规则集对所述编程语言进行类型检查，

其中，所述多个类型检查规则集包括对应于强类型检查的一个类型检查规则集、对应于弱类型检查的一个类型检查规则集、以及对应于表示类型检查的一个类型检查规则集。

2．如权利要求1所述的方法，其特征在于，所述编程语言的特征描述了所述语言的类型系统。

3．如权利要求1所述的方法，其特征在于，对所述编程语言进行类型检查包括对所述编程语言的多个中间表示的每一个进行类型检查。

4．如权利要求3所述的方法，其特征在于，所选择的一个或多个类型检查规则集对每一中间表示是不同的。

5．如权利要求1所述的方法，其特征在于，所述编程语言包括指示所述编程语言的元素可以是多个类型中的一个类型的类型。

6．如权利要求5所述的方法，其特征在于，所述一个或多个类型检查规则集包含用于对指示所述编程语言的元素可以是多个类型中的一个类型的类型进行类型检查的规则。

7．如权利要求1所述的方法，其特征在于，所述一个或多个类型检查规则集包括分层格式的多个规则。

8．如权利要求1所述的方法，其特征在于，所述表示类型检查允许对所述编程语言的元素丢弃类型信息。

9．如权利要求1所述的方法，其特征在于，所述弱类型检查允许类型强制转换。

10．一种用于对以多种源语言创作的源代码进行类型检查的类型检查系统，包括：

多个类型检查规则集；以及

类型检查器，其中，所述类型检查器是编译器的一部分，所述类型检查器选

择一个或多个类型检查规则集以在多个中间表示的每一个中间表示处应用于所述源代码，

其中，所述多个类型检查规则集包括对应于强类型检查的一个类型检查规则集、对应于弱类型检查的一个类型检查规则集、以及对应于表示类型检查的一个类型检查规则集。

11. 如权利要求10所述的系统，其特征在于，所述多个类型检查规则集的每一个对应于特定的源语言。

12. 如权利要求10所述的系统，其特征在于，所述多个类型检查规则集的每一个对应于特定的类型检查强度。

13. 一种在编译器中对被表示为类型化的中间表示的源代码进行类型检查的方法，包括：

基于所述类型化的中间表示从不同的类型检查规则集中选择一个类型检查规则集，其中选择一个类型检查规则集是基于编译过程中的一系列阶段中的哪一阶段产生了所述类型化的中间表示；以及

基于所选择的类型检查规则集对所述类型化的中间表示进行类型检查，

其中，所述多个类型检查规则集包括对应于强类型检查的一个类型检查规则集、对应于弱类型检查的一个类型检查规则集、以及对应于表示类型检查的一个类型检查规则集。

14. 如权利要求13所述的方法，其特征在于，选择类型检查规则集是基于从其中产生所述类型化的中间表示的源语言。

15. 如权利要求13所述的方法，其特征在于，选择类型检查规则集是基于处理器体系结构。

16. 如权利要求13所述的方法，其特征在于，选择类型检查规则集是基于所述类型化的中间表示是表示验证的还是未验证的代码。

17. 如权利要求13所述的方法，其特征在于，选择类型检查规则集是基于所述类型化的中间表示是表示类型安全的代码还是非类型安全的代码。

18. 如权利要求13所述的方法，其特征在于，选择类型检查规则集是基于所述类型化的中间表示是表示类型化的代码还是非类型化的代码。

19. 如权利要求13所述的方法，其特征在于，选择类型检查规则集是基于所述类型化的中间表示是表示强类型化的代码还是弱类型化的代码。

在编译过程中表示和检查程序组件的一致性的可扩展类型系统

技术领域

本发明涉及类型系统，尤其涉及可扩展到新的和经更新的编程语言的类型系统。

背景

类型系统是在编程语言中用于协助检测和防止运行时出错的系统。如果编程语言包含一组对于诸如变量、函数等对象声明的类型，且这些类型在以该语言编写的程序的编译期间对照一组规则来检查，则该编程语言是“类型化的”。如果以类型化语言编写的源代码违反了类型规则之一，则确定为编译器错误。

用于编译器的类型化的中间语言在过去几年中经受了研究团体的大量研究。它们增强了编译器的可靠性和健壮性，以及提供了一种跟踪和检查无用信息收集器所需的信息的系统方法。其概念是具有一种在其上附加了类型且可以用类似于源程序的类型检查的方式来进行类型检查的中间表示。然而，类型化的中间语言更难以实现，因为在编译过程中把表示项的类型变得明确是必要的。

如果类型化的中间语言必须表示多个不同的高级编程语言，则它甚至更难以实现。不同的语言不仅具有不同的原语操作和类型，而且高级编程语言具有不同的类型化级别。例如，诸如汇编语言等某些语言一般是非类型化的。换言之，它们没有类型系统。在类型化的语言中，某些是强类型化的，而其它是更松散地类型化的。例如，C++是一般被认为松类型化的语言，而 ML 或 Pascal 被认为是强类型化的语言。此外，松类型化的某些语言具有较小的语言子集，这些子集允许程序内的大部分代码段是强类型化的，而其它代码段是松类型化的。例如，C#和.NET 中使用的微软中间语言（MSIL）允许这一类型化。因此，用于表示这些高级语言的任一种的类型化的中间语言必须能够表示不同的类型强度。同样，这一类型化的中间语言的类型系统必须能够根据进行类型检查的代码的特征来实现不同的规则。

当贯穿整个编译过程降级类型化的中间语言时，会引发另一问题。语言的降级指的是将语言的形式从诸如程序员所编写的较高级形式改为诸如中间语言等较

低级形式的过程。语言然后可从中间语言进一步降级到诸如机器相关的本机代码等接近于计算机执行的级别。为在编译过程中对被降级到不同级别的中间语言进行类型检查，必须对每一表示使用一组不同的规则。

创建类型化的中间语言的尝试通常无法解决上述问题。例如，Cedilla System 的 Special J 编译器使用了一种类型化的中间语言。然而，这一编译器对于 Java 源语言是专用的，且因此不需要处理例如具有非类型安全代码的多种语言。另外，该编译器仅使用一组规则用于类型检查，且因此不能用于多个编译级别。在研究团体中，类型化的中间语言往往对源语言是高度专用的，因此难以为多个编译阶段工程实现（以及设计类型）。

概述

提供了一种用于检查各种形式的中间语言的一致性的类型表示、类型检查器、方法和编译器。具体地，类型化的中间语言适用于表示以多种（完全不同的）源语言编写的程序，这些源语言包括类型化的和非类型化的语言、松和强类型化的语言、以及带有和不带有无用信息收集的语言。另外，该类型检查器体系结构是可扩展的，以处置具有不同类型和原语操作的新语言。类型表示、类型检查器、方法和编译器包括各个方面。各个方面可分开且独立地使用，或者各个方面可以按各种组合和子组合来使用。

在一方面，提供了一种在编译器中对编程语言进行类型检查的方法。取一个或多个规则集作为对类型检查器的输入，类型检查器基于多个准则中的任何一个，或两个或多个的组合选择规则集中的一个或多个。在它们之中的是编译阶段、源语言、体系结构以及进行类型检查的语言中存在的类型化级别。语言然后使用所选择的一个或多个规则集进行类型检查。

在另一方面，提供了一种带有类型检查器的编译器，类型检查器基于多个准则中的任何一个，或两个或多个的组合构造一个或多个规则集。规则集可包括对应于强类型检查的一个规则集、对应于弱类型检查的一个规则集、对应于表示类型检查的一个规则集。弱规则集可允许类型化中的更多灵活性，诸如允许类型强制转换，而表示规则集可允许在中间程序表示的各部分中丢弃的类型信息。

在另一方面，提供了一种用于构造用于检查程序的中间表示的一致性的多个规则的编程接口。检查中间表示的一致性可包括向类型检查器提供多个规则，类型检查器基于预定准则向一个中间表示应用第一组规则，以及向另一中间表示应用第

二组规则。

当参考附图阅读以下详细描述，可以清楚这些和其它方面。

附图简述

图 1 是通用编译过程的流程图。

图 2 是示出将源代码语句转换成高级表示，然后转换成机器相关的低级表示的表清单。

图 3 是示出用于在各个编译阶段对类型化的中间语言进行类型检查的编译器系统的一个实施例的数据流程图。

图 4 是用于编译器系统的类型检查器的框图。

图 5 是用于选择要由类型检查器应用的规则集的一种可能的过程的流程图。

图 6 是示出类型之间的分层关系的有向图。

图 7 是示出将类型添加到类型之间的分层关系的有向图。

图 8 是用于对照类型检查系统中的类型规则检查指令的方法的流程图。

图 9 是担当类型检查系统的一个实施例的操作环境的计算机系统的示例的框图。

详细描述

提供了用于检查各种形式的中间语言的一致性的类型表示、类型检查器和编译器。类型检查器和编译器允许根据编程组件和/或编译阶段的源语言使用不同的类型和类型检查规则。例如，可能期望将高级优化器应用于以各种语言编写的程序。这些语言可具有不同的原语类型和原语操作。例如，一种语言可以包含用于复杂算术的类型和操作，而另一种语言可包含对计算机图形专用的类型和操作。通过允许由不同的类型系统来参数化中间表示，优化器可用于具有不同原语类型和操作的语言。另一示例可包括其中某些组件以强类型化的语言子集来编写，而其它组件以非类型安全的完全语言来编写的程序。期望对于第一组组件进行更多的检错。这可以通过对不同的组件使用不同的类型检查规则来实现。又一示例是在编译期间丢弃类型信息。类型检查器和编译器可允许在较晚的阶段丢弃类型信息，而可在较早的阶段强迫维护精确的信息。这可以通过对不同的编译阶段结合不同的类型检查规则使用未知类型来实现。

图 1 示出了使用具有不同降级级别的类型化的中间语言来表示多种不同的源

语言的系统的通用编译过程。源代码 100-106 是以四种不同的源语言来编写的，这些源语言可以是类型化或不是类型化的，且具有不同的类型强度级别。例如，以 C#编写的源代码 100 将比以 C++编写的源代码 106 更强地类型化。源代码首先由读取器 108 处理并进入系统。源语言然后被翻译成类型化的中间语言的高级中间表示（HIR）。HIR 然后可在框 110 被可任选地分析并优化。HIR 然后被翻译成类型化的中间语言的中级中间表示（MIR）。该表示低于 HIR，但是仍是机器无关的。在这一点上，MIR 可如框 112 所示被可任选地分析并优化。MIR 然后由代码生成在框 114 翻译成类型化的中间语言的机器相关的低级表示（LIR）。LIR 然后可在框 116 被可任选地分析并优化，并在框 118 被提供给发放器。发放器将以表示读入系统的原始源代码的多种格式 120-126 中的一种输出代码。贯穿整个过程，完成过程所必需的数据被储存在某一形式的持久存储器 128 中。

由此，该编译过程包括将中间语言指令从一个表示级别翻译成另一表示级别。例如，图 2 示出将源代码语句转换成 HIR，以及将 HIR 转换成机器相关的 LIR。源代码语句 200 可以用多种高级编程语言来编写。这些语言被设计成允许程序员以容易理解的方式读写代码。由此，允许程序员使用如“+”等字符用于加法，并允许使用更强大的形式，诸如添加如语句 200 中所示的两个以上操作数。

语句 202-206 是表示同一功能的语句 200 的 HIR 表示，但是以更接近于可由机器理解且仍是体系结构无关的格式来完成。语句 202 使用“ADD”命令来将第一和第二变量相加，并将结果赋值给第一临时变量 t1。语句 204 然后使用另一“ADD”命令将 t1 与第三变量相加，并将结果赋值给第二临时变量 t2。语句 206 然后使用“ASSIGN”指令将值 t2 赋值给结果变量 z。

语句 208-121 是语句 202-206 的中间语言的 LIR。语句 208 使用对 x86 体系结构专用的 add 指令将储存在指定寄存器处的两个变量的值相加，并将结果储存在被赋值给临时变量 t1 的寄存器中。语句 210 使用对 x86 体系结构专用的 add 指令来将值 t1 和储存在指定的寄存器处的第三变量相加，并将结果储存在被赋值给 t2 的指定寄存器（EAX）中。语句 212 然后使用对 x86 体系结构专用的 move 指令将储存在 EAX 中的值移至输出变量 z。

为实现类型检查，类型化的中间语言包含显式或隐式表达的类型表示。显式类型表达式是直接在表示中声明的。例如，语句：

int a;

显式地将变量“a”定义为类型 int。类型表示可通过为某些代码语句定义一默认类

型来隐式地表达。例如，如果函数的默认返回类型是 int，则语句：

f_start();

将声明没有任何形参并返回类型为 int 的值的函数 f_start。

适用于多个表示级别处的多个编程语言的类型化的中间语言的类型表示的一个实施例在附录 A 中示出。应当注意，这仅是众多可能的实施例中的一个。

参考附录 A，在类型类分层结构中定义了多个类型表示，使得各种语言的类型系统可由类型化的中间语言来表示。抽象基类对于所有类型被定义为“Phx::Type”。该基类可包含例如“sizekind”中用于各种类型的大小信息，这些类型诸如实际、符号或未知（或可变）类型。基类也可包含“typekind”，以指定类型分类。另外，可提供外部类型，作为包装外部定义的类型的抽象类型，以提供从类型化的中间语言到原始源代码的反向映射。

在基类之下，被定义为“Phx::PtrType”的类可表示指针类型。也可定义各种指针。例如，受管的、已收集无用信息的指针（指向已收集无用信息的对象内的位置）；受管的、未收集无用信息的指针（指向未收集无用信息的对象内的位置）；非受管指针（诸如在以例如 C++编写的代码中找到的）；引用指针（指向已收集无用信息的对象的基类）；以及空指针。

在分层结构中的同一级别处，被定义为“Phx::ContainerType”的类可表示容器类型，诸如包含内部成员的类型。内部成员可具有作用域、方法以及其它类型。被定义为“Phx::FuncType”的类可表示函数类型，包括任何必要的调用约定、形参列表和返回类型列表。同样，被定义为“Phx::UnmgdArrayType”的类可表示非受管的数组类型。在分层结构中的“Phx::ContainerType”下还可定义四个类。被定义为“Phx::ClassType”的类可表示类类型，被定义为“Phx::StructType”可表示结构类型，被定义为“Phx::InterfaceType”的类可表示接口类型，而被定义为“Phx::EnumType”的类可表示枚举类型。在分层结构中的“Phx:ClassType”下，被定义为“Phx::MgdArrayType”的另一个类可表示受管的数组类型。

在附录 A 所示的表示中，类“primtype”被定义为结构类型的特殊实例。“primtype”可包括诸如 int、float、unknown（未知）、void、condition code（条件代码）、unsigned int（无符号整数）、xint 等各种类型。这些表示可在类型化的中间语言的 HIR 和 LIR 中使用。

另外，目标专用的原语类型可被包括在类型表示中。某些语言具有复杂算术类型，如果类型系统知道它们，则它们可被有效地处理。例如，考虑“MMX”指

令。这一指令是被构建到某些版本的 x86 处理器中用于支持多媒体和通信数据类型上的单指令/多数据操作的一组外部指令集中的一个。类型系统可被定制成以对类型表示的最小改变来识别并使用这些指令。

附录 A 中所示的类型表示的实施例也包括“unknown”类型，它可表示任何类型，并可任选地具有与其相关联的大小。大小是值的机器表示的大小。unknown 类型允许编译器通过将类型信息从特定类型改变为 unknown 类型以受控的方式来丢弃类型信息。它允许编译器生成取决于所操纵的值的大小的代码，即使在类型是 unknown 的情况下。其它类型可使用 unknown 类型，因此 unknown 类型也允许部分类型信息的表示（其中某些但并非所有信息是已知的）。

例如，假定指向 int 类型的指针。在某一降级阶段，可能期望丢弃类型信息 int。unknown 类型允许编译器用 unknown 类型来替换 int 类型。类型检查器然后无需检查所关注的指针是否指向正确的类型。它本质上进行这样的冒险：即所指向的值将以在运行时不会不利地影响程序功能的方式来处理。

使用 unknown 类型的另一示例是用于为函数定义类型。如果调用具有指向 unknwon 的类型指针的形参的函数，其中形参先前已将类型指针指向 int，则编译器必需相信传递了正确的类型。解除指针引用的结果可以已知或不是已知为 int；然而，它可用作 int。一个更复杂的示例是在虚拟函数调用从高级到低级中间表示的转换期间中间临时变量的引入。虚拟表（v 表）在面向对象的语言中广泛用于实现虚拟调用。低级中间表示中作出虚拟函数调用的第一步是从存储器中取出对象的第一字段。第一字段包含指向 v 表的指针。取数据的结果然后被赋值给临时变量。构造临时变量的类型（表示指向 v 表的指针的类型，其中 v 表可具有许多字段）可以是复杂且难以表示的。相反，编译器可简单地向中间临时变量赋值“指向 unknown 的指针”。由此，对 unknown 类型的使用简化了较晚阶段的编译，在这些阶段中，保存详细类型信息是不必要的，或可能对编译器实现器表现出重大负担。

图 3 示出了用于在编译的各阶段对类型化的中间语言表示进行类型检查，并因此示出了在各个降低级别上对类型化的中间语言进行类型检查的编译器系统的一个实施例。源代码 300 表示各种源语言中的任一种。源代码 300 被翻译成类型化的中间语言的 HIR 302。通过这样做，源语言的类型表示被翻译成对于类型化的中间语言的内部类型表示。

如对于图 1 和 2 所解释的 HIR 贯穿整个编译过程被降级。为该图示的目的，示出了高（HIR）302、中（MIR）304 以及低（LIR）306 级表示。然而，该实施

例不限于此。可对任意数量的编译阶段进行类型检查。

每一表示级别处的中间语言可以由类型检查器 308 进行类型检查。类型检查器 308 实现了用于向编译过程的每一阶段，并因此向中间语言的每一表示应用一个或多个规则集 310 的算法或过程。规则集 310 是为语言的各种属性，诸如源语言、编译阶段、何种类型化强度等设计的一组规则。

例如，假定源代码 300 包含以 C++编程语言创作的代码。C++源代码 300 首先被翻译成类型化的中间语言的 HIR 302。如有所需，在这一点上，类型检查器 308 可与 HIR 302 交互，以确定任何数量的属性。这些属性可包括编译阶段（HIR）、存在的源代码的类型（C++）、语言是否类型化（是）、是松类型化还是强类型化（松）等等。基于这些属性，类型检查器可选择一适当的规则集。一旦选择了一个规则集，类型检查器依照该规则集对 HIR 进行类型检查。一旦 HIR 被降级到 MIR 或 LIR，将再次访问属性，并且相同或不同的规则集可以是适当的。

在一个实施例中，这些类型检查规则集可被提供给类型检查器。一个规则集可对应于“强”类型检查，诸如对 C#或 MSIL 的类型检查是期望的。另一规则集可对应于“弱”类型检查，它可以是比“强”类型检查更松的类型检查。例如，弱类型检查规则集可以准许类型转换。类型强制转换是使一种类型的变量作为另一种类型来行动以供单次使用。例如，可使类型 int 的变量如 char（字符）那样来行动。以下代码使用了类型转换来打印字母“P”。

```
int a;
a=80;
cout<<(char)a;
```

由此，即使“a”被定义为类型 int，并被赋值 80，但由于类型转换，cout 语句将变量“a”作为类型 char 来对待，并因此显示“P”（ASCII 值 80）而非 80。

最后，一个规则集可对应于“表示”检查。“表示”检查可允许中间程序表示的各部分中丢弃的类型信息，诸如通过使用 unknown 类型，并可包括指示这一类型信息何时被丢弃时或何时可用 unknown 类型替代另一类型时的规则。例如，返回类型 Void 的值的函数的结果可以禁止被赋值给 unknown 类型的变量。

另外，可在单个编译阶段使用一个以上规则集。例如，假定源代码 300 包含单个语言，但是包含强类型化的某些段和松类型化的某些段。类型检查器可对某些强类型化段的 HIR 使用一个规则集，并对松类型化的代码段使用另一规则集。

图 4 是在类似于图 3 所描述的编译器系统中使用的类型检查器的框图。类型

检查器 400 可接受对应于不同源语言和/或不同编译阶段的任意数量的规则集作为输入。在图 4 中，将四个规则集 402-408 提供给类型检查器 400。规则集 402 表示用于具有强类型化语言的 HIR 的规则集，规则集 404 表示用于具有弱类型化语言的 HIR 的规则集，规则集 406 表示用于具有无类型化语言的 HIR 的规则集，而规则集 408 表示用于 LIR 的规则集。程序模块 410 表示具有 HIR 形式的强类型化的语言，而程序模块 412 表示在被降级到 LIR 之后的程序模块 410。

类型检查器 400 基于正进行类型检查的程序模块的属性选择适当的规则集，并使用结合的过程或算法将所选择的规则集应用于程序模块。例如，类型检查器 400 可选择规则集 402（表示用于具有强类型化语言的 HIR 的规则集），以对程序模块 410（表示具有 HIR 形式的强类型化的语言）进行类型检查。随后，类型检查器 400 然后可选择规则集 408（表示用于 LIR 的规则集），以对程序模块 412（表示具有 LIR 形式的强类型化的语言）进行类型检查。

图 5 是用于选择由类型检查器应用的规则集的过程的一个可能实施例的流程图。在框 500，类型检查器读入源代码的类型化中间表示的一个段，并且必需选择用于类型检查的规则集。判别框 502 确定类型化的中间语言是 HIR、MIR 还是 LIR。

如果是 HIR 或 MIR，则处理判别 504 以确定原始源代码是松类型化还是强类型化的。如果是松类型化的，则处理框 506 以选择对应于弱类型检查的规则集。如果是强类型化的，则处理框 508 以选择对应于强类型检查的规则集。

如果是 LIR，则处理判别框 510 以选择对应于表示类型检查的规则集。应当注意，图 5 仅是一个实施例。可选择对应于且基于不同属性的任意数量的规则集。

所描述的类型检查系统的规则集可被容易地扩展到完全新的语言，并也可被扩展到现有语言的新特征。例如，如果引入新的语言，则只需为该新语言创作新规则集。由于规则集与类型检查器或编译器本身是分离的，且被设计成接受规则集作为单独的实体，因此用于新语言的新规则集可被分发，而无需重新分发或更新现有的类型检查系统或编译器。同样，如果新特征被添加到现有语言，诸如向 C++添加 XML 支持，则在各编译阶段处对应于 C++的规则集可被任意地动态重新配置以处理新特征。再一次，无需更新或分发新的核心系统。

规则集也可允许对类型约束。例如，当类从另一类继承时是否允许对特定类型使用子类型化可以是在规则中描述的一个约束。另一约束可以是加框约束，诸如可能期望指示数据可被转换成包含数据的虚拟表。其它约束可包括大小约束，或指示相同原语类型的必要性的原语类型约束。与规则集的任何其它部分一样，新约束

可如所需地添加。

由类型检查器使用的规则集可以通过用于创作规则集的应用程序编程接口来构造。该应用程序可构造规则，使得规则集在类型原语的分层结构中用分配给类型化中间语言的个别指令的规则来表示。分层结构可以用类型图的形式来表示，类型图明确地表达了与特定程序模块或编译单元有关的类型的各种元素。诸如符号和操作等 IR 元素与类型系统的元素相关联。类型图节点描述了原语，并构造了类型及其关系，诸如组件、嵌套类型、函数签名、接口类型、分层结构元素以及诸如源名和对模块/汇编外部类型元素的引用等其它信息。

简单类型规则的一个示例如下：

ADD

N=add n, n

为本示例的目的，假定 I 是 signed integer（带符号整数）类型，U 是 unsigned integer 类型，X 是任一 integer 类型，F 是 float，N 是上述的任一种。图 6 示出了这些类型之间的分层关系。类型 N 在分层结构的顶部。类型 F 和 X 从类型 N 向下分支以形成分层结构的后续层次。最后，类型 U 和 I 从 X 类型向下分支以形成分层结构的最低层。由此，对于“ADD”中间语言指令，依照该规则，仅类型 N 或分层结构中的较低层可由 add 指令来处理，且操作数在分层结构中必须不高于结果。例如，两个整数可相加，以产生整数（I=ADD i, i），或者一个整数和一个浮点数可相加以产生浮点数（F=ADD i, f）。然而，浮点数和整数不能相加来产生整数（I=ADD i, f）。

将类型原语表示为分层结构允许容易地改变规则集。在过去，类型规则通常在编程上使用源代码来表达。例如，类型检查器可包含实现类型检查器规则的大量开关语句。由此，改变规则要求修改类型检查器的源代码。然而，分层的规则集提供了更容易的可扩展性。考虑先前用于 ADD 指令的规则。如果开发者希望添加一个类型，例如复杂类型 C，则可简单地在分层结构的 N 类型下方添加，如图 7 所示，且用于 ADD 指令的规则无需改变以如所需地运作。

用于在类型检查系统中对照类型规则检查指令的一种方法在图 8 中示出。首先，对框 800 处理以在句法上检查指令。由此，考虑 806 处的指令，类型检查器确保依照用于 ADD 指令的类型规则存在正确数量的源和目标表达式（例如，在这一情况下，有 2 个源表达式以及 1 个目标表达式）。每一表达式（以及子表达式）可具有其上的中间表示的显式类型。在框 802，类型检查器然后实际地验证 e1、e2

以及 foo(e3)的显式类型符合用于 ADD 指令的类型规则。在框 804，类型检查器在必要时遍历子层次以进一步对指令进行类型检查。例如，类型检查器可检查表达式 e1、e2 和 foo(e3)与其显式类型相一致。例如，类型检查器可检查 foo 具有函数类型。它可检查函数类型的结果类型与 foo(e3)上的显式类型相同。它还可检查有单个形参类型，以及类型 e3 与该类型相匹配。这确保对 e3 的调用的类型与类型规则相匹配。

图 9 示出了担当用于类型检查系统的一个实施例的操作环境的计算机系统的示例。计算机系统包括个人计算机 920，包括处理单元 921、系统存储器 922 以及将包括系统存储器的各类系统组件耦合至处理单元 921 的系统总线 923。系统总线可以包括若干种总线结构类型的任一种，包括存储器总线或存储器控制器、外围总线以及使用各类总线体系结构的局部总线，仅举几个例子，这类体系结构诸如 PCI、VESA、微通道（MCA）、ISA 和 EISA。系统存储器包括只读存储器（ROM）924 和随机存取存储器（RAM）925。基本输入/输出系统（BIOS）926，包含如在启动时协助在计算机 920 内的元件之间传输信息的基本例程，可储存在 ROM 924 中。个人计算机 920 也可包括硬盘驱动器 927，例如用于对可移动磁盘 929 进行读写的磁盘驱动器 928，以及例如用于读 CD-ROM 或对其它光介质进行读写的光盘驱动器 930。硬盘驱动器 927、磁盘驱动器 928 以及光盘驱动器 930 分别通过硬盘驱动器接口 932、磁盘驱动器接口 933 和光盘驱动器接口 939 连接至系统总线 923。驱动器及其相关联的计算机可读介质为个人计算机 920 提供了数据、数据结构、计算机可执行指令（诸如动态链接库等程序模块以及可执行文件）等的非易失性存储。尽管对计算机可读介质的描述指的是硬盘、可移动磁盘以及 CD，然而也可以包括计算机可读的其它类型的介质，包括盒式磁带、闪存卡、数字多功能盘、Bernoulli 盒式磁盘、RAM、ROM 等等。

多个程序模块可储存在驱动器和 RAM 925 中，包括操作系统 935、一个或多个应用程序 936、其它程序模块 937 以及程序数据 938。用户可以通过键盘 940 和定点设备（诸如鼠标 942）向计算机 920 输入命令和信息。其它输入设备（未示出）可包括麦克风、操纵杆、游戏垫、圆盘式卫星天线、扫描仪等等。这些和其它输入设备通常通过耦合至系统总线的串行端口接口 949 连接到处理单元 921，但也可通过其它接口连接，如并行端口、游戏端口或通用串行总线（USB）。监视器 947 或另一显示设备也通过接口，如显示控制器或视频适配器 948 连接到系统总线 923。除监视器之外，个人计算机通常包括其它外围输出设备（未示出），如扬声器和打

印机。

个人计算机 920 可以使用到一个或多个远程计算机，如远程计算机 949 的逻辑连接在网络化环境中操作。远程计算机 949 可以是服务器、路由器、网络 PC、对等设备或其它常见的网络节点，并通常包括许多或所有相对于个人计算机 920 所描述的元件，尽管在图 9 中仅示出了存储器存储设备 950。图 9 描述的逻辑连接包括局域网（LAN）951 和广域网（WAN）952。这类网络环境常见于办公室、企业范围计算机网络、内联网以及因特网。

当在 LAN 网络环境中使用时，计算机 920 通过网络接口或适配器 953 连接至局域网 951。当在 WAN 网络环境中使用时，个人计算机 920 通常包括调制解调器 954 或用于通过广域网 952，如因特网建立通信的其它装置。调制解调器 954 可以是内置或外置的，通过串行端口接口 946 连接至系统总线 923。在网络化环境中，相对于个人计算机 920 所描述的程序模块或其部分可储存在远程存储器存储设备中。可以理解，示出的网络连接是示例性的，也可以使用在计算机之间建立通信链路的其它手段。

示出并描述了所示实施例的原理之后，本领域的技术人员可以明白，可以在排列和细节上修改这些实施例而不脱离这些原理。

例如，此处的一个实施例描述了一个或多个规则集，它们可被提供给类型检查器或编译器，使得编译器或类型检查器基于进行类型检查的语言和/或编译阶段来选择一个或多个规则集以对语言进行类型检查。然而，作为替换，可将单个规则集提供给类型检查器或编译器，使得编译器或类型检查器在运行时静态或动态地、基于进行类型检查的语言和/或编译阶段从该单个规则集中构造一个或多个规则子集

鉴于许多可能的实施例，可以认识到，所示实施例仅包括示例，且不应当被认为是限制本发明的范围。相反，本发明由所附权利要求书来定义。因此，要求保护落入所附权利要求书的范围内的所有这样的实施例作为本发明。

附录 A

```
//-----------------------------------------------------------------------------//
// 描述：
//
//   IR 类型
//
//
//    类型类分层结构                    引入的描述和原语属性
//    --------------------          ------------------------------------------
//    Phx::Type                        - 类型的抽象基类
//        Phx::PtrType                 - 指针类型
//        Phx::ContainerType           - 容器类型（具有成员的类型）
//                Phx::ClassType       - 类类型
//                    Phx::MgdArrayType - 受管数组类型
//                Phx::StructType      - 结构类型
//                Phx::InterfaceType   - 接口类型
//                Phx::EnumType        - 枚举类型
//        Phx::FuncType                - 函数类型
//                                       属性：  ArgTypes, ReturnType
//
//        Phx::UnmgdArrayType          - 受管数组
//                                       属性：Dim, Referent
//
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
//
// 描述：
//
//   用于 IR 类型的基类
//
//-----------------------------------------------------------------------------

__abstract __public __gc
class Type : public Phx::Object
{

public:

    // 用于比较类型的函数

    virtual Boolean IsAssignable(Phx::Type * srcType);
    virtual Boolean IsEqual(Phx::Type * type);

public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::TypeKind,          TypeKind,       typeKind);
    DEFINE_GET_PROPERTY(Phx::SizeKind,          SizeKind,       sizeKind);
    DEFINE_GET_PROPERTY(Phx::BitSize,           BitSize,        bitSize);
    DEFINE_GET_PROPERTY(Symbols::ConstSym *,    ConstSym,       constSym);
    DEFINE_GET_PROPERTY(Phx::ExternalType *,    ExternalType, externalType);
    DEFINE_GET_PROPERTY(Phx::PrimTypeKindNum, PrimTypeKind, primTypeKind);
    GET_PROPERTY(Phx::TypeSystem *, TypeSystem);
```

```
protected:

    // 受保护域

    Phx::TypeKind           typeKind;        // type classification
    Phx::SizeKind           sizeKind;        // size classification
    Phx::BitSize            bitSize;         // size in bits when constant
    Symbols::ConstSym *     constSym;        // size in bits when symbolic
    Phx::PrimTypeKindNum primTypeKind;
    Phx::ExternalType *     externalType;    // optionally null
};

//-----------------------------------------------------------------------------
//
// 描述:
//
//    容器类型 - 用于具有成员的类型的抽象类
//
//-----------------------------------------------------------------------------

__abstract __public __gc
class ContainerType : public QuantifiedType, public IScope
{

    DEFINE_PROPERTY(Symbols::FieldSym *, FieldSymList, fieldSymList);

private:

    // 私有域

    Symbols::FieldSym * fieldSymList;

};

//-----------------------------------------------------------------------------
//
// 描述:
//
//     类容器类型
//
//-----------------------------------------------------------------------------

__public __gc
class ClassType : public ContainerType
{

public:

    // 公有静态构造函数

    static Phx::ClassType * New
    (
        Phx::TypeSystem *       typeSystem,
        Phx::BitSize            bitSize,
        Phx::ExternalType *     externalType
    );
```

```
public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::Type *, UnboxedType,           unboxedType);
    DEFINE_PROPERTY(       ClassType *, ExtendsClassType, extendsClassType);

    DEFINE_GET_PROPERTY(Phx::Collections::InterfaceTypeList *,
        ExplicitlyImplements, explicitlyImplements);

protected:

    // 受保护属性

    DEFINE_SET_PROPERTY(Phx::Collections::InterfaceTypeList *,
        ExplicitlyImplements, explicitlyImplements);

private:

    // 私有域

    Phx::Type *                                  unboxedType;
    Phx::ClassType *                             extendsClassType;
    Phx::Collections::InterfaceTypeList * explicitlyImplements;
};

//-----------------------------------------------------------------------------
//
// 类：StructType
//
// 描述：
//
//      结构类型
//
//
//-----------------------------------------------------------------------------

__public __gc
class StructType : public ContainerType
{

public:

    // 公有静态构造函数

    static Phx::StructType * New
    (
        Phx::TypeSystem *          typeSystem,
        Phx::BitSize               bitSize,
        Phx::ExternalType *        externalType
    );

public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::ClassType *, BoxedType, boxedType);
```

```
private:

    // 私有域

  Phx::ClassType * boxedType;
};

//------------------------------------------------------------------------------
//
// 类：PrimType
//
// 描述：
//
//      原语类型
//
//------------------------------------------------------------------------------

__public __gc
class PrimType : public StructType
{

public:

    // 公有静态构造函数

    static Phx::PrimType *
    New
    (
        Phx::TypeSystem *           typeSystem,
        Phx::PrimTypeKindNum        primTypeKind,
        Phx::BitSize                bitSize,
        Phx::ExternalType *         externalType
    );

public:

    // 公有方法

    static Phx::PrimType *
    GetScratch
    (
        Phx::PrimType * type,
        PrimTypeKindNum kind,
        Phx::BitSize bitSize
    );

    Phx::PrimType * GetResized
    (
        Phx::BitSize bitSize
    );

public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::TypeSystem *,   TypeSystem, typeSystem);
```

```
private:

    // 私有域

    Phx::TypeSystem * typeSystem;
};

//------------------------------------------------------------------------------
//
// 类：InterfaceType
//
// 描述：
//
//      接口类型
//
//------------------------------------------------------------------------------

__public __gc
class InterfaceType : public ContainerType
{

public:

    // 公有静态构造函数

    static Phx::InterfaceType * New
    (
        Phx::TypeSystem *           typeSystem,
        Phx::ExternalType *         externalType
    );

public:

    // 公有属性

    DEFINE_PROPERTY(        Phx::ClassType *, ExtendsClassType, extendsClassType);
    DEFINE_GET_PROPERTY(Phx::Collections::InterfaceTypeList *,
        ExplicitlyImplements, explicitlyImplements);

protected:

    // 受保护属性

    DEFINE_SET_PROPERTY(Phx::Collections::InterfaceTypeList *,
        ExplicitlyImplements, explicitlyImplements);

private:

    // 私有域

    Phx::ClassType *                          extendsClassType;
    Phx::Collections::InterfaceTypeList * explicitlyImplements;
};

//------------------------------------------------------------------------------
//
// 类：EnumType
//
```

```
// 描述、：
//
//   枚举类型
//
//-----------------------------------------------------------------------------

__public __gc
class EnumType : public ContainerType
{

public:

    // 公有静态构造函数

    static Phx::EnumType * New
    (
        Phx::TypeSystem *          typeSystem,
        Phx::ExternalType *        externalType,
        Phx::Type *                underLyingType
    );

public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::ClassType *, BoxedType, boxedType);
    DEFINE_GET_PROPERTY(Phx::Type *, UnderlyingType, underlyingType);

private:

    // 私有域

    Phx::Type * underlyingType;
    Phx::ClassType * boxedType;
};

//-----------------------------------------------------------------------------
//
// 类：MgdArrayType
//
// 描述：
//
//      受管数组类型
//
//
//-----------------------------------------------------------------------------

__public __gc
class MgdArrayType : public ClassType
{

public:

    // 公有静态构造函数

    static Phx::MgdArrayType * New
    (
        Phx::TypeSystem *          typeSystem,
```

```
        Phx::ExternalType *            externalType,
        Phx::Type *                      elementType
    );

public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::Type *,            ElementType,    elementType);

private:

    // 私有域

    Phx::Type *    elementType;
};

//-----------------------------------------------------------------------------
//
// 类：UnmgdArrayType
//
// 描述：
//
//      非受管数组类型
//
//
//-----------------------------------------------------------------------------

__public __gc
class UnmgdArrayType : public Type
{

public:

    // 公有静态构造函数

    static Phx::UnmgdArrayType * New
    (
        Phx::TypeSystem *               typeSystem,
        Phx::BitSize                    bitSize,
        Phx::ExternalType *             externalType,
        Phx::Type *                      referentType
    );

public:

    // 公有属性

    DEFINE_GET_PROPERTY(int,               Dim,                 dim);
    DEFINE_GET_PROPERTY(Phx::Type *, Referent,            referent);

private:

    // 私有域

    int               dim;
    Phx::Type *    referent;
};
```

```
//-----------------------------------------------------------------------------
//
// 描述:
//
//      指针类型
//
//
//-----------------------------------------------------------------------------

__public __gc
class PtrType : public Type
{

public:

    // 公有静态构造函数

    static Phx::PtrType * New
    (
        Phx::TypeSystem *           typeSystem,
        Phx::PtrTypeKind            ptrTypeKind,
        Phx::BitSize                bitSize,
        Phx::Type *                 referent,
        Phx::ExternalType *         externalType
    );

    //  不带所指向的类型的构造函数

    static Phx::PtrType * New
    (
        Phx::TypeSystem *           typeSystem,
        Phx::PtrTypeKind            ptrTypeKind,
        Phx::BitSize                bitSize,
        Phx::ExternalType *         externalType
    );

public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::PtrTypeKind,     PtrTypeKind,     ptrTypeKind);
    DEFINE_GET_PROPERTY(Phx::Type *,          Referent,        referent);

private:

    // 私有域

    Phx::PtrTypeKind        ptrTypeKind;
    Phx::Type *             referent;
};

//-----------------------------------------------------------------------------
//
// 枚举: CallingConvention
//
// 描述:
```

```
//
//    表示调用约定类型的初步枚举
//
//----------------------------------------------------------------------------

BEGIN_ENUM(CallingConventionKind)
{
    _Illegal = 0,
    CLRCall,
    CDecl,
    StdCall,
    ThisCall,
    FastCall
}
END_ENUM(CallingConventionKind);

//----------------------------------------------------------------------------
//
// 类：FuncType
//
// 描述
//
//    函数类型
//
//
//----------------------------------------------------------------------------

__public __gc
class FuncType : public QuantifiedType
{

public:

    // 公有静态构造函数

public:

    // 公有方法

    Int32               CountArgs();
    Int32               CountRets();
    Int32               CountArgsForInstr();
    Int32               CountRetsForInstr();
    Int32               CountUserDefinedArgs();
    Int32               CountUserDefinedRets();
    Phx::FuncArg * GetNthArgFuncArg(Int32 index);
    Phx::FuncArg * GetNthRetFuncArg(Int32 index);
    Phx::FuncArg * GetNthArgFuncArgForInstr(Int32 index);
    Phx::FuncArg * GetNthRetFuncArgForInstr(Int32 index);
    Phx::FuncArg * GetNthUserDefinedArgFuncArg(Int32 index);
    Phx::FuncArg * GetNthUserDefinedRetFuncArg(Int32 index);
    Phx::Type *      GetNthArgType(Int32 index);
    Phx::Type *      GetNthRetType(Int32 index);
    Phx::Type *      GetNthArgTypeForInstr(Int32 index);
    Phx::Type *      GetNthRetTypeForInstr(Int32 index);
    Phx::Type *      GetNthUserDefinedArgType(Int32 index);
    Phx::Type *      GetNthUserDefinedRetType(Int32 index);
```

```
public:

    // 公有属性

    DEFINE_GET_PROPERTY(Phx::CallingConventionKind,                CallingConvention,
callingConvention);
    GET_PROPERTY(Phx::Type *,      RetType);

    // 如果该函数类型具有省略的函数形参则为真

    GET_PROPERTY(Boolean,          IsVarArgs);
    GET_PROPERTY(Boolean,          IsInstanceMethod);
    GET_PROPERTY(Boolean,          IsClrCall);
    GET_PROPERTY(Boolean,          IsCDecl);
    GET_PROPERTY(Boolean,          IsStdCall);
    GET_PROPERTY(Boolean,          IsThisCall);
    GET_PROPERTY(Boolean,          IsFastCall);

    // 如果该函数具有返回值则不为真

    GET_PROPERTY(Boolean,          ReturnsVoid);

protected:

    // 受保护域

    Phx::CallingConventionKind     callingConvention;
    Phx::FuncArg *                 argFuncArgs;
    Phx::FuncArg *                 retFuncArgs;
};

//-----------------------------------------------------------------------------
//
// 描述:
//
//     编译期间使用的全局类型系统
//
//
//-----------------------------------------------------------------------------

__public __gc
class TypeSystem : public Phx::Object
{

public:

    // 公有静态构造函数

    static TypeSystem *
    New
    (
        Phx::BitSize        regIntBitSize,
        Phx::BitSize        nativeIntBitSize,
        Phx::BitSize        nativePtrBitSize,
        Phx::BitSize        nativeFloatBitSize,
    );

public:
```

```
    // 公有方法

    void Add(Type * type);

public:

    // 创建的类型的列表

    DEFINE_GET_PROPERTY(Phx::Type *, AllTypes,            allTypes);

private:

    // 系统中所有类型的列表

    Phx::Type * allTypes;
};

//--------------------------------------------------------------------------
//
// 描述:
//
//    描述类型的各种枚举
//
//--------------------------------------------------------------------------

// 用于 IR 类型的类和值类型

// 不同种类的类型

BEGIN_ENUM(TypeKind)
{
    _Illegal = 0,
    Class,
    Struct,
    Interface,
    Enum,
    MgdArray,
    UnmgdArray,
    Ptr,
    Func,
    Variable,
    Quantifier,
    Application,
    TypedRef,
}
END_ENUM(TypeKind);

// 不同种类的类型大小

BEGIN_ENUM(SizeKind)
{
    _Illegal = 0,
    Constant,
    Symbolic,
    Unknown
}
END_ENUM(SizeKind);
```

```
// 不同类型的指针

BEGIN_ENUM(PtrTypeKind)
{
    _Illegal = 0,
    ObjPtr,                 // 指向整个对象的__gc 指针
    MgdPtr,                 // 指向内部对象的__gc 指针
    UnmgdPtr,               // __nogc 指针
    NullPtr,                // 不指向任何内容的指针
    _NumPtrTypeKinds
}
END_ENUM(PtrTypeKind);
}
```

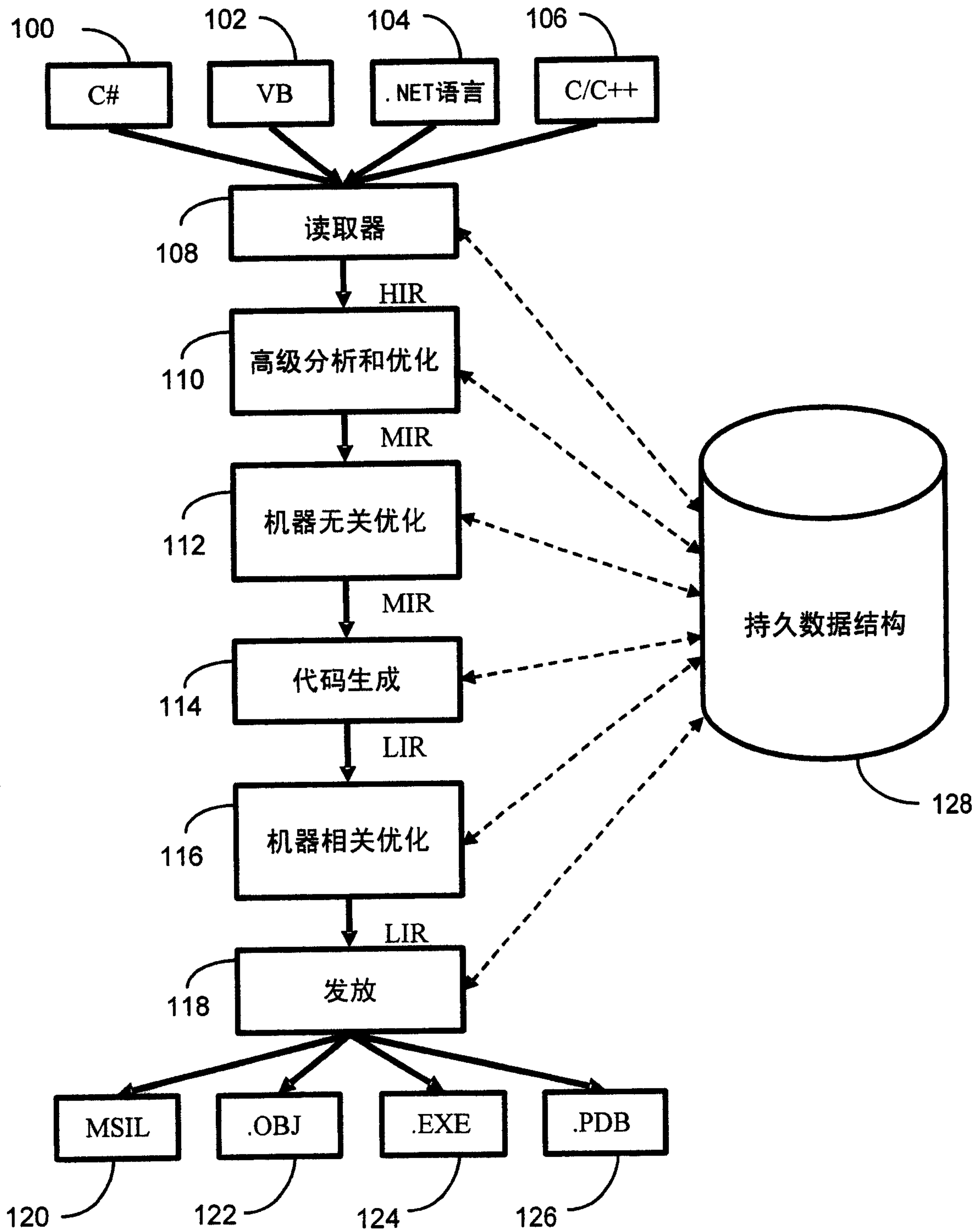


图 1

源:

```
z = a + b + c;- 200
```

HIR:

```
t1 = ADD     a, b- 202
t2 = ADD    t1, e 204
  z = ASSIGN t2 - 206
```

LIR:

```
t1(EAX), cc = x86add a(EAX), b(EDX) - 208
 t2(EAX), cc = x86add t1(EAX), c(EBX) - 210
             z = x86mov t2(eax)   - 212
```

图 2

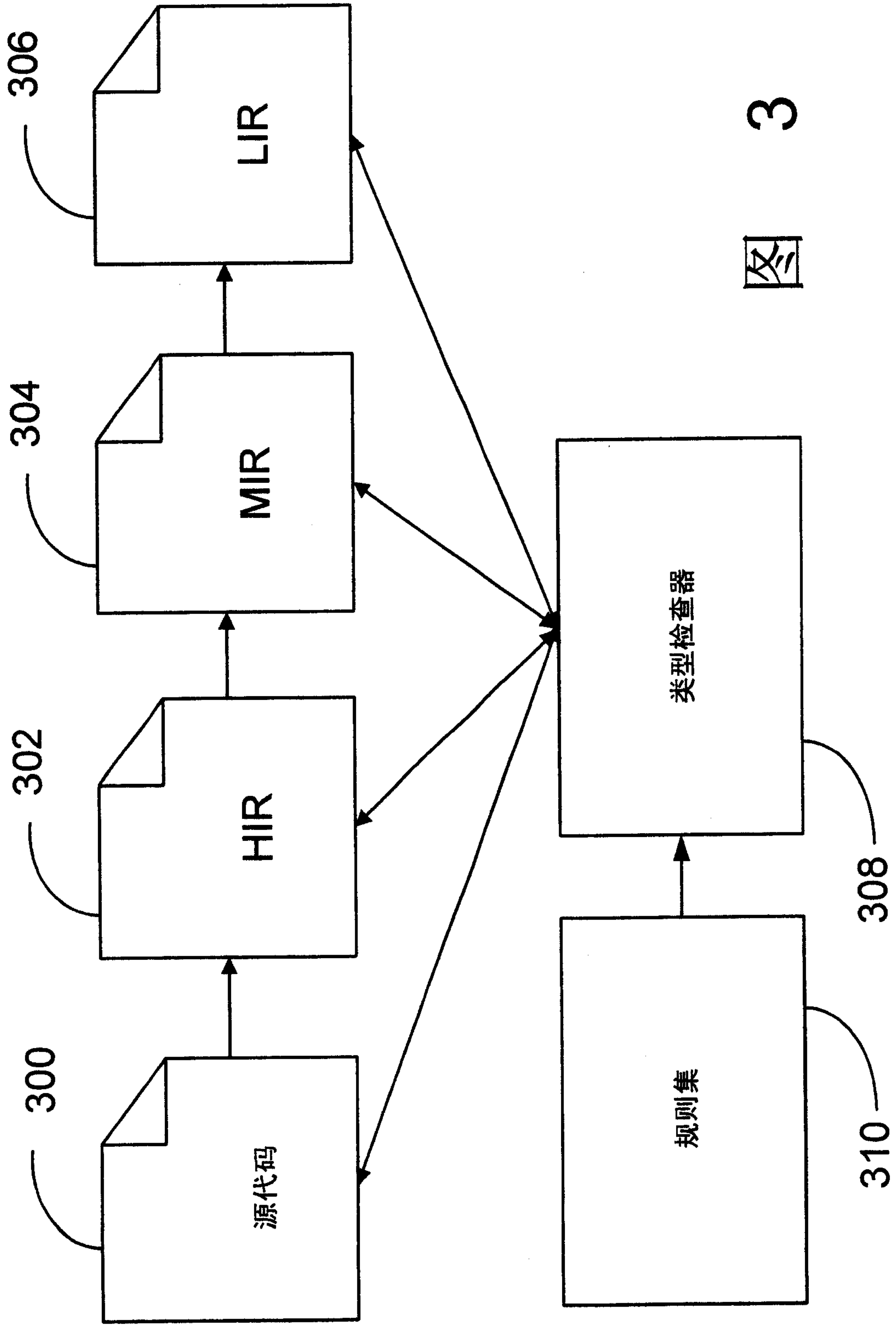


图 3

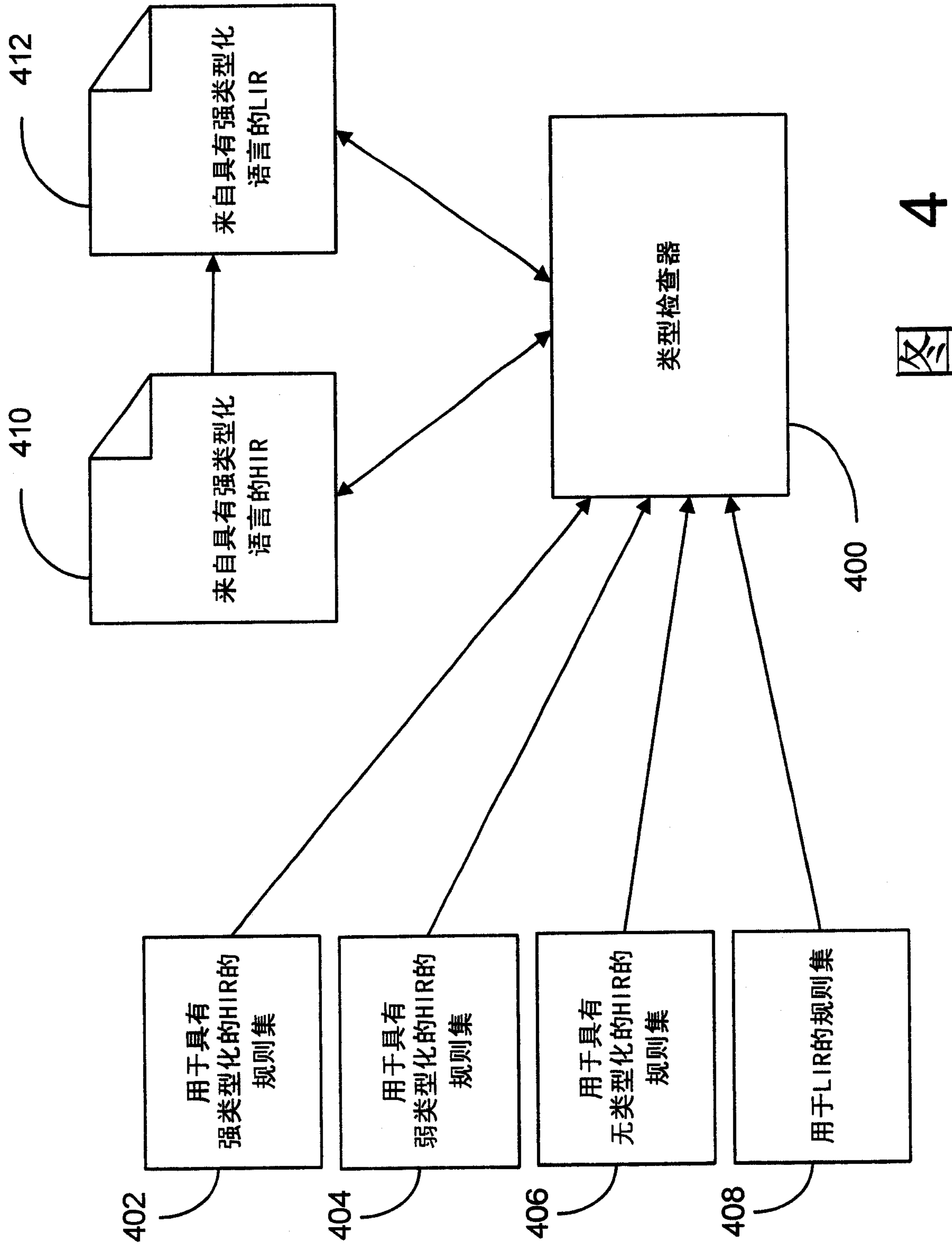


图 4

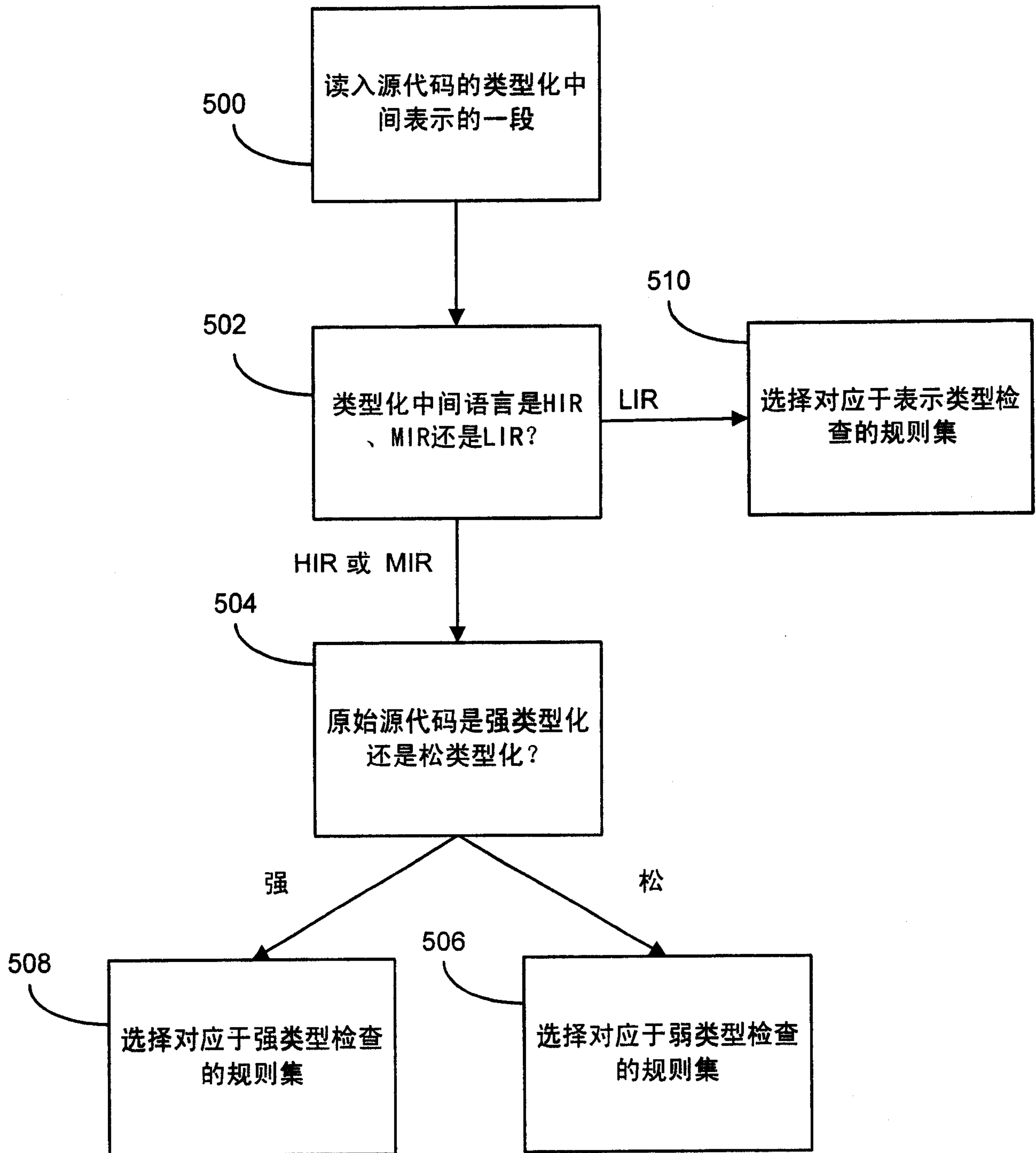


图 5

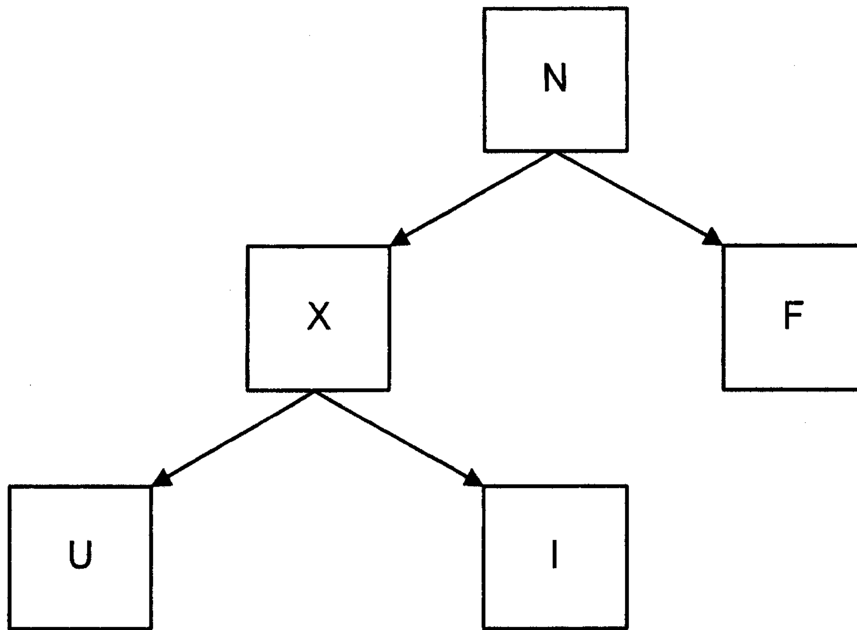


图 6

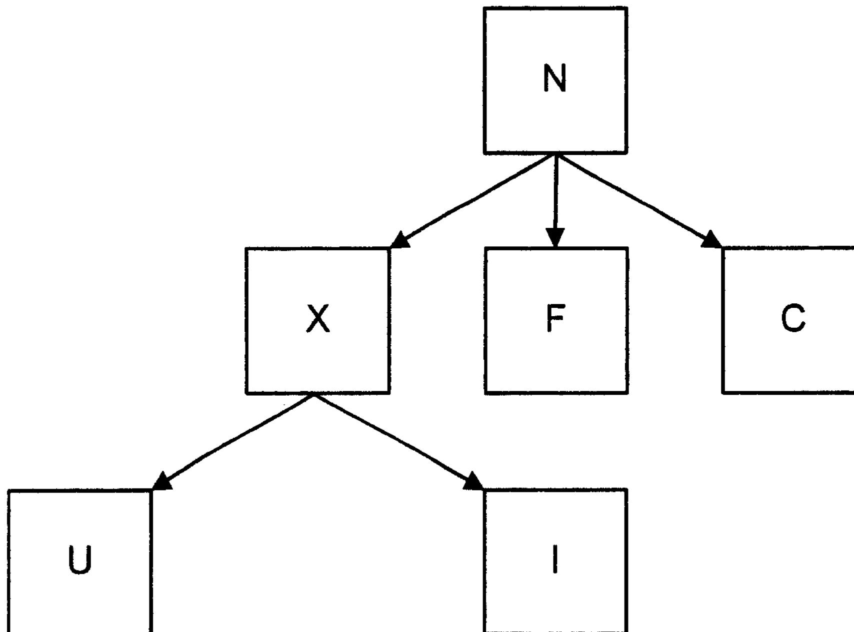


图 7

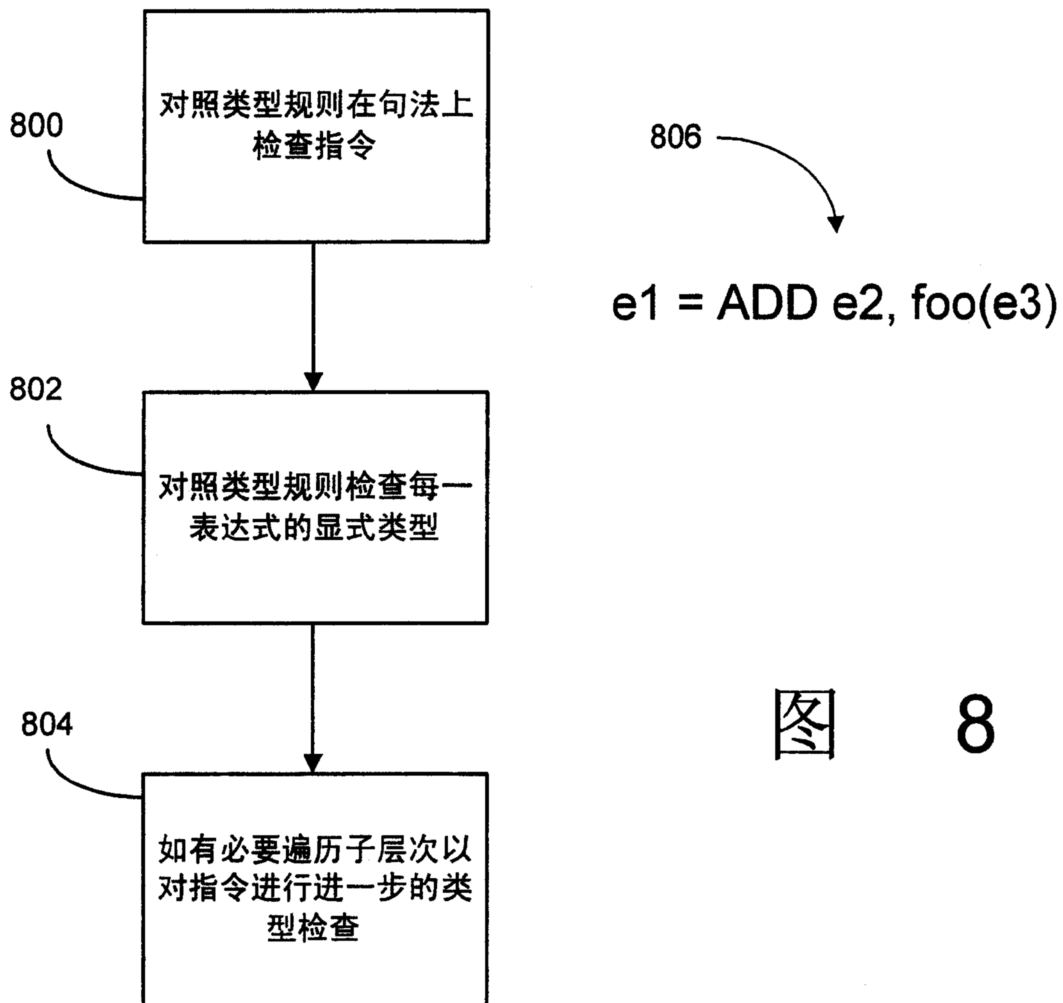


图 8

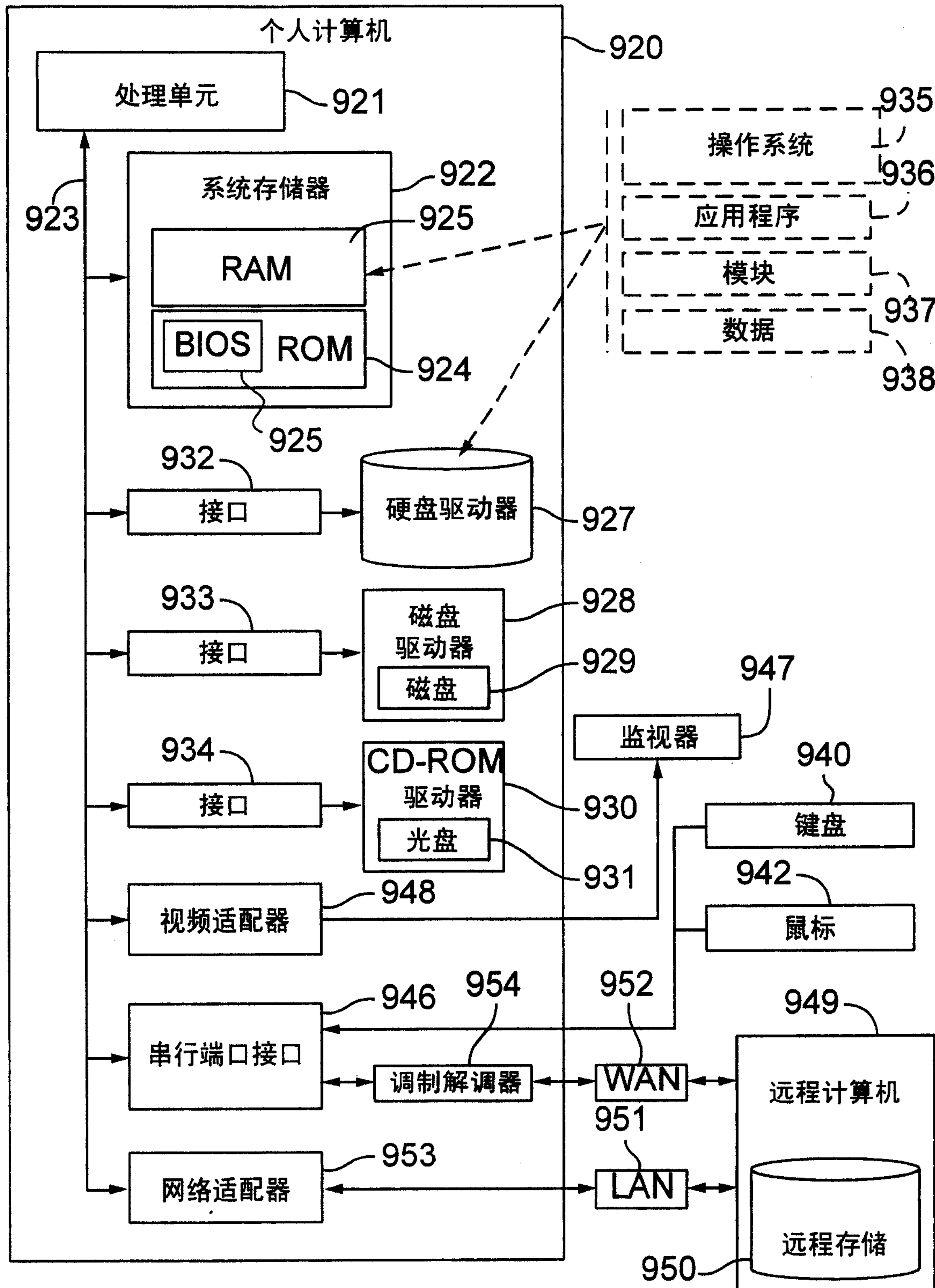


图 9