(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 102365621 B
(45)授权公告日 2015.08.26

(21)申请号 201080015827.X
(22)申请日 2010.03.26
(30)优先权数据
12/383,732 2009.03.27 US
(85)PCT国际申请进入国家阶段日
2011.09.26
(86)PCT国际申请的申请数据
PCT/US2010/000929 2010.03.26
(87)PCT国际申请的公布数据
WO2010/110925 EN 2010.09.30
(73)专利权人 奥普塔姆软件股份有限公司
地址 美国加利福尼亚州
(72)发明人 D·R·谢里登
(74)专利代理机构 中国专利代理(香港)有限公司 72001
代理人 姜甜 马永利
(51)Int.Cl.
G06F 9/44(2006.01)
G06F 17/20(2006.01)
G06F 9/06(2006.01)

(56)对比文件
CN 101336413 A,2008.12.31,
CN 101336413 A,2008.12.31,
US 2006/0080647 A1,2006.04.13,
US 2006117302 A1,2006.06.01,

审查员 梁静静

权利要求书2页 说明书12页

(54)发明名称

使用嵌入式解释器类型和变量的基于解释器的程序语言翻译器

(57)摘要

编程语言被扩展成具有嵌入式解释类型(EIT),这些EIT定义在翻译时要解析的对象和变量。具有这些EIT之一的类型的变量或数据元被称为嵌入式解释变量(EIV)。在翻译时解释(即,执行)包含EIV的控制结构。

CN 102365621 B

1. 一种用于将计算机程序自动翻译成计算机可执行形式的方法，所述方法包括：

提供用输入计算机语言表达的输入计算机程序，其中所述输入计算机语言包括在运行时要评估的语言元素或结构的第一数据类型集合、以及在翻译时要评估的语言元素或结构的第二数据类型集合，并且其中所述输入计算机程序包括具有所述第一数据类型集合中的类型的至少一个元素或结构并且包括具有所述第二数据类型集合中的类型的至少一个元素或结构；

将所述输入计算机程序自动翻译成计算机可执行输出表示，其中在所述自动翻译期间评估具有所述第二数据类型集合中的类型的至少一个元素或结构；以及

提供所述计算机可执行输出表示作为输出，

其中，根据语言元素或结构的数据类型来确定在翻译时对所述语言元素或结构的评估以及在运行时对所述语言元素或结构的评估，将所述输入计算机程序自动翻译成计算机可执行输出表示包括将所述输入计算机程序翻译成中间表示，并且其中所述中间表示只包括在运行时要评估的结构，以及其中

运算符的翻译时评估包括：

确定向所述运算符提供的自变量类型，其中至少一种所述自变量类型在所述自动翻译期间被评估为类型值；

提供具有与所确定的自变量类型相一致的输入自变量类型的所述运算符的版本。

2. 如权利要求 1 所述的方法，其特征在于，所述输入计算机语言具有预定和固定的语法，并且其中语言结构的语义含义不取决于所述语言结构是具有所述第一数据类型集合还是第二数据类型集合中的类型。

3. 如权利要求 1 所述的方法，其特征在于，在翻译期间通过包括在编译器中的嵌入式解释器来评估具有所述第二数据类型集合中的类型的至少一个元素或结构。

4. 如权利要求 3 所述的方法，其特征在于，所述编译器包括将其输入流中的每个名称默认地作为文字处理的定义输入模块，并且其中所述定义输入模块在翻译具有所述第二数据类型集合中的类型的结构期间调用所述嵌入式解释器。

5. 如权利要求 4 所述的方法，其特征在于，所述嵌入式解释器和定义输入模块可彼此递归地调用。

6. 如权利要求 1 所述的方法，其特征在于，所述输入计算机程序包括具有所述第二数据类型集合中的类型的所述输入计算机语言的一个或多个结构，所述结构在翻译期间提供相应数据元或结构的延迟例示。

7. 如权利要求 1 所述的方法，其特征在于，具有所述第二数据类型集合中的类型的至少一个元素或结构包括从“if-then”、“if-then-else”、“for”、“do-while”和“switch-case”组成的组中选择的控制结构。

8. 一种用于将计算机程序自动翻译成计算机可执行形式的系统，所述系统包括：

第一装置，用于提供用输入计算机语言表达的输入计算机程序，其中所述输入计算机语言包括在运行时要评估的语言元素或结构的第一数据类型集合、以及在翻译时要评估的语言元素或结构的第二数据类型集合，并且其中所述输入计算机程序包括具有所述第一数据类型集合中的类型的至少一个元素或结构并且包括具有所述第二数据类型集合中的类型的至少一个元素或结构；

第二装置，用于将所述输入计算机程序自动翻译成计算机可执行输出表示，其中在所述自动翻译期间评估具有所述第二数据类型集合中的类型的至少一个元素或结构；以及

第三装置，用于提供所述计算机可执行输出表示作为输出，

其中，根据语言元素或结构的数据类型来确定在翻译时对所述语言元素或结构的评估以及在运行时对所述语言元素或结构的评估，将所述输入计算机程序自动翻译成计算机可执行输出表示包括将所述输入计算机程序翻译成中间表示，并且其中所述中间表示只包括在运行时要评估的结构，以及其中

运算符的翻译时评估包括：

确定向所述运算符提供的自变量类型，其中至少一种所述自变量类型在所述自动翻译期间被评估为类型值；

提供具有与所确定的自变量类型相一致的输入自变量类型的所述运算符的版本。

使用嵌入式解释器类型和变量的基于解释器的程序语言翻译器

发明领域

[0001] 本发明涉及编程语言的自动翻译。

[0002] 背景

[0003] 编程语言的基本模型是根据语法所规范的符号集合，该基本模型根据语言的语义规则被解析并翻译成可执行形式。以这种形式，在每个输入结构指向输出表示中的特定结构的意义上，语言是“文字”。即，过程规范产生可执行过程规范，并且结构或类型或类规范产生用于访问存储器中的该结构或类型或类的元素的存储器模板（偏移集合）。

[0004] 存在避免重复规范并识别公共处理序列和数据结构的各种方式。具体而言，出现在多个位置的公共子处理序列可通过对公共子例程的调用来替代，该公共子例程执行那个公共处理序列。出现在多个表达式中的公共子表达式还可被析出为赋值给本地变量的表达式或者析出为返回子表达式的值的单独函数。类似地，公共数据结构可用公共类型 / 类或结构规范来指定，其中数据成员按照该公共类型来指定。面向对象的语言允许使用这些不同类型继承的基本类型来识别并指定不同类型之间的共性，从而有效地共享基本类型数据成员以及成员函数的定义和实现。

[0005] 然而，可产生包括通过这些共享机制不能处理的差异的逻辑共性。具体而言，一数据结构实现的两个规范可以是相同的，但要存储在该数据结构中的对象的类型除外。例如，散列表的两种实现可只在各自可存储的对象的类型方面有所不同。类似地，两种不同的计算平台的处理序列可以是相同的，而只在正传递给调用的常量的名称方面有所不同。

[0006] 有可能通过在运行时执行包括自动运行时检查的动作来处理一些这样的情况。例如，常量可被改变成变量，其中该变量具有在运行时根据目标平台自动设置的值。类似地，散列表可具有基本类型（可能是所谓的“空”基本类型指针）并且执行实际上存储在表中以确保类型安全的类型的运行时检查。然而，这种方法招致只在运行时发现问题的风险。它还招致执行实际上在翻译时已知的并且由此逻辑上是不必要的各个方面的运行时检查的成本。最后，对于诸如 Verilog 生成和嵌入式软件之类的一些情况，执行运行时检查可能是不可行的。例如，路由器实现可能需要端口模块的 N 次实现，每次实现根据系统参数而稍有不同，但在翻译时而非运行时被静态地例示。

[0007] 这些问题的一种早期解决方法是添加所谓的宏预处理器，该宏预处理器（预）处理语言输入并且随后关于所得经变换输入调用语言翻译器本身。该宏处理器允许指定由宏处理器所识别的结构，从而使它修改输入源作为文本变换。例如，C/C++ 预处理器是提供命名常量、宏（内联过程）、条件编译和文件蕴含的宏处理器。然而，这种方法在实践中有若干问题。具体而言，文本变换可产生非法结果和很难的出错消息、以及引入更难于捉摸的错误。它还可使软件调试困难，因为输入源没有准确地反映适当地向语言翻译器提供了什么。

[0008] 最近，类型参数化的类型和过程已被包括在语言中，因此只在所使用的类型或常量方面有所不同的公共代码被指定一次，随后针对每个特定类型例示。C++ 模板机制是一个示例。然而，C++ 类型名没有语义特性。它只是一个绑定到类的类型参数。具体而言，C++

类型的例示可导致只在链接时发现而不是由翻译器发现的错误。C# 和 Java 的类属是相似的，不同之处在于它们允许为这些参数指定一些约束或不变量。

[0009] 另一种方法是支持对语言中的基本语言结构的重新定义，该重新定义经常被称为元编程。例如，语言可允许程序员指定对诸如函数调用运算符之类的基本运算符的修订定义。然而，这种方法有效地创建了新语言，从而使其他程序员难以理解软件。它还未解决诸如条件编译、文件蕴含等关键的实践需求。

[0010] 因此，提供用于在翻译时处理逻辑共性的机制将是本领域的进步，这些机制相对于传统方法提供了通用性和易用性的经改进组合。

[0011] 概述

[0012] 在本方法中，编程语言被扩展成具有嵌入式解释类型 (EIT)，这些 EIT 定义在翻译时要解析的对象和变量。具有这些 EIT 之一的类型的变量或数据元被称为嵌入式解释变量 (EIV)。在翻译时解释（即，执行）包含 EIV 的控制结构。

[0013] 为了更好地理解这种方法的重要性，考虑与计算机编程语言有关的“类型”的常规定义是有帮助的。通常，类型被定义为编程语言的数据元的标签或指示，其指示 :a) 数据元的值的容许范围 ;以及 b) 数据元的容许操作 / 运算符。例如，整数数据类型可具有从 -32,768 到 32,767 的容许范围并且具有包括诸如 +、-、* 和 / 之类的算术运算符在内的容许操作。作为另一个示例，串数据类型可被定义为字节阵列，这些字节根据所编码的字符表示字符并且具有诸如级联、子串提取等容许操作。试图对串（或整数的级联）执行算术运算导致在翻译时容易捕捉的类型不匹配错误，这在实践中是相当有益的。

[0014] 另一种理解本方法的重要性的方式是将这种方法与诸如宏预处理和使用例如 C++ 中的模板之类的常规方法进行比较和对比。宏文本替换本质上是“不知道类型的”进程。例如，如果命名为“xyz”的参数通过文本替换被设为特定值（例如，20），则在文本替换期间不（或不可以）进行类型检查。由此，如果声明变量“xyz”具有除整数以外的类型，则文本替换可引入类型错误。

[0015] 类似地，模板元编程也不依赖于作为类型的模板，或者不依赖于在模板例示之前对模板参数的类型检查。模板是哪个代码可通过例示生成所依据的模式，并且它是通过模板例示提供的代码的具有类型（或者可以是类型）的元素。模板支持通过编译器的分段解释的形式，其中类模板提供了对参数化类型的延迟或分段指定。（部分）模板专门化提供了实现选择和循环（在递归方面）的手段，由此提供了完整的语言实用工具。C++ 模板编程用类型来有效地计算，从而使用循环（以及终止循环）的递归和专门化以及嵌套的 enum（枚举）值来有效地指定一个值。然而，编程可能是棘手的、低效的、并且导致非显而易见的错误信息。此外，模板本身不是类型，因为模板不确定哪些运算符可合法地适用于模板、并且不确定模板的容许范围。

[0016] 本工作可被视为基于类型的常规概念的扩展。因此，从这一点起，类型被定义为关于编程语言的数据元的标签或指示，其指示 :a) 数据元的值的容许范围 ;b) 数据元的容许操作 / 运算符 ;以及 c) 在运行时或在翻译时是否要评估数据元。在翻译时评估的类型是上述的 EIT。

[0017] 使用类型来确定在翻译时评估哪些结构提供了若干重要的优点。

[0018] 第一，在设计时未确定或者不可确定而在翻译时已知的各个方面可被解析成文字

/ 静态表示，从而允许翻译时检查并且避免运行时绑定的开销。具体而言，支持静态例示。

[0019] 第二，通过在翻译时的评估期间“在语言翻译器内”操作，翻译器访问语言的符号、从而允许使用所定义的常量，并且访问诸如类型、属性和约束的集合之类的其他对象。这允许更容易地标识句法和语义问题。

[0020] 第三，因为EIT受限于语言的句法元素，所以所得代码能够更容易被其他程序员理解，并且可确保与外部经编译模块的兼容性。同样，不需要对语言语法的进一步复杂化，因为每个EIV结构使用相同的语法。例如，诸如a: = b+c之类的赋值语句的含义不取决于a、b和c是否是在翻译时评估的EIV。

[0021] 根据上述原理的自动程序翻译可如下进行。输入计算机程序可用输入计算机语言提供，其中输入计算机语言提供两个数据类型集合。第一类型集合用于在运行时要评估的语言元素或结构（即，常规类型）。第二类型集合用于在翻译期间要评估的语言元素或结构（即，EIT）。输入计算机程序包括具有第一类型集合中的类型的至少一个元素或结构，以及具有第二类型集合中的类型的至少一个元素或结构。在自动翻译期间，评估输入计算机程序中的具有第二集合中的类型的任何元素或结构（即，具有EIT类型的EIV）。提供自动翻译的输入计算机程序的计算机可执行表示作为输出（例如，作为可执行机器代码）。

[0022] 优选地，输入计算机语言具有预定和固定的语法，以使语言结构的语义含义不取决于该结构的类型是在第一数据类型集合中还是在第二数据类型集合中。

[0023] 在一个实施例中，解释器嵌在编译器内，其中嵌入式解释器负责EIV的翻译时评估。术语“嵌入式解释类型”和“嵌入式解释变量”从这种可能性得出。

[0024] 在一些实施例中，编译器包括定义输入模块(DIM)，DIM具有将其输入流中的每个名称默认地作为文字处理的特性。DIM可在翻译EIT类型结构期间调用嵌入式解释器。通常，翻译器将符号（例如，猫）解释为要评估的变量名，并且要求附加句法（例如，作为“猫”的引用）强迫翻译器将该符号视为文字（即，串文字“猫”）。然而，在DIM中，默认是替换地将未引用的符号（例如，猫）解释为相应的文字（例如，“猫”），而不是要评估的变量名。DIM和嵌入式解释器可彼此递归地调用。

[0025] 在一些情况下，上述自动翻译可包括将输入计算机程序翻译成只具有在运行时要评估的结构的中间表示。该中间表示可以是高级（即，人类可读）语言或低级（即，机器可读）语言。

[0026] 在程序翻译期间，不必确保所有EIV根据先前在对翻译器的输入流中所定义的量来表达。通过提供供延迟例示用的EIT结构，有可能根据稍后要在翻译期间定义的量来定义EIV。一种用于提供这种延迟例示的方法是提供用于EIT/EIV结构的模板，该模板稍后可在翻译时处理期间被例示。

[0027] 整个范围的控制结构可用于与EIT结构一起使用，包括但不限于“if-then”、“if-then-else”、“for”、“do-while”和“switch-case”。

[0028] 通过EIT/EIV方法所提供的灵活性的示例是以下用于静态运算符超载的方法。输入计算机语言的运算符可被赋予EIT类型，并且这些运算符的翻译时评估可包括确定向运算符提供的自变量类型、以及提供其输入自变量类型与实际所提供的自变量类型相一致的运算符的版本。以这种方式，避免了运行类型的类型检查，并且不需要程序员对输入程序中的运算符的每个实例进行类型检查。

[0029] 详细描述

[0030] 如上所指示的，EIT 提供了指定在翻译时解析（和必须解析）的源代码输入中的“变量”的能力，从而允许在翻译时选择常量值、代码和声明序列以及类型，有效地参数化类似软件单元中的差异，允许它们以常见的方式来表达。

[0031] 各种类型的 EIT 在实践中是令人感兴趣的。EIT 可与诸如“整数”之类的常规类型相对应，但指示它需要解释并且由此在翻译时具有可确定的值。作为另一个示例，EIT 可与用于翻译器的输入流的类型相对应。EIT 可与编程语言语法中的节点的类型和其附属类型之一相对应。例如，如果语言包括“类型”结构，则可存在与该类型结构相对应的 EIT 以及与用于在嵌套的（子）类型集合上迭代的迭代器类型相对应的 EIT。

[0032] 在一个实施例中，翻译器可被构造成包括解释器组件，该解释器组件执行关于程序的内部表示 (IR) 的指令。这些指令包括查询和修改该 IR 的手段。例如，一个实施例可包含解释器指令或命令，诸如：

[0033] 从“Bar”定义类型“Foo”；

[0034] 以定义从现有类型“Bar”得出（或继承）的命名为“Foo”的新类型。由此，该解释器语言的变量是指 IR 中的对象，并且名称值是引用的文字值。嵌在翻译器所处理的编程语言中的 EIT 是通过该解释器来实现的类型，从而允许解释处理被指定并从该编程语言中调用，不管它是被编译还是被解释。

[0035] 在该示例语句中，引用名称“Foo”和“and”以指示它们是文字、并且特殊“定义”的关键字指示这是与翻译时评估有关的定义，而不是只在翻译时翻译（而不评估）的正常程序语句。优选地，根据本方法的翻译利用如以下所述的定义输入模块，以更有效地执行翻译时评估。

[0036] 定义输入模块处理

[0037] 解释器优选支持根据编程语言句法中的指定输入流来读取、解析和更新 IR 的操作。该操作通过解释器调用定义输入模块处理 (DIM) 来实现，该 DIM 读取指定输入流并且相应地更新 IR。相对于正常解释器模式，该模式默认为定义模式并且将每个名称默认为文字，从而消除对引用的需要（如在前述的示例中）。特殊的句法值可由保留关键字指示，如本领域所公知的。

[0038] 在该 DIM 输入模式中，程序可用常规编程语言模型来指定，在常规编程语言模型中名称默认为文字并且结构是可定义的或可声明的。相反，使用更常规的通用解释器方法（其中会需要大量的定义）具有以下缺点：a) 引用所有文字元素（诸如如上“Foo”的定义所示的类型名称等）所引起的句法混乱；以及 b) 与使用以上定义一样不得不指定每个定义结构所引起的关键字混乱。

[0039] 在一个实施例中，IR 中的类型节点被标记为 EIT。类似地，声明为其类型是 EIT 类型的变量被考虑并且被标记为 EIV。由此，IR 中的类型的 EIV 是那个结构的变量，而不是特定文字实例。换言之，节点可以是表示类型的类型变量，该类型变量在翻译期间的某一点处被绑定到文字类型，而不是特定文字类型名称。例如，类型 NumericType 的 EIV NType（其中 NumericType 是 EIT）可在翻译期间被绑定到诸如整数、浮点数、双精度或复数之类的文字类型名称。

[0040] 当 DIM 遇到结构中的 EIV、作为解析某输入的一部分时，它可将该结构的指示写到

在调用时向它提供的预定义 EIT 对象。这个动作调用解释器处理该结构。解释器处理该结构，从而可能对它进行重写并传递给 DIM（的新实例）来处理、并且随后更新预定义 EIT 对象以指示通过调用 DIM 的后续动作（如果有的话）。

[0041] 通过遇到 EIV 所触发的这个回调提供了一种将要由解释器处理的结构有效地嵌入这种编程语言输入的方式，诸如文件蕴含（即，递归导入）和条件翻译。

[0042] 在一个实施例中，DIM 还可在模板模式中调用，在这种情况下，它解析输入流、从而接受未被绑定到特定值的 EIT 和 EIV，并且创建稍后可调用的经解析结构（例如，EIT/EIV 模板）的延迟例示内部表示。实际上，这为嵌入式解释器提供了宏能力。

[0043] 因此，为了调用解释器，DIM 的输入形式无需偏离底层语言语法。此外，DIM 可用于向解释器提供简明形式的宏能力。而且，IR 的 EIT/EIV 元素可以直接方式用于解释器，只要它们是用于语言本身的相同 IR 的元素。

[0044] EIT 规范

[0045] 在一个实施例中，EIT 可在诸如 Interp（解释器的命名空间）之类的保留命名空间中指定。例如

[0046] Interp::String fileName =″ input.tac″ ;

[0047] EIV 随后可通过将它指定为该保留命名空间中的这些类型之一用经解析语言表示来定义。在一个实施例中，该命名空间包括与翻译器 IR 语法中的类型相对应的类型、相关联的迭代器和附属类型、诸如串、整数之类的基本类型，并且还包括专用类型和预定义 EIV。例如，在一个实施例中，Interp::translatorIn 对应于翻译器的输入流。

[0048] 这种方法利用了编程语言被指定并实现为语言语法的事实，该编程语言根据与诸如类型或类、过程、变量、表达式等不同语言结构相对应的非终结符号来指定。

[0049] 被定义为 EIT 的派生类型的类型也是 EIT。它可用附加运算符、属性和特性来定义。在这种情况下，这种派生类型 EIT 的变量可只被赋予与这些要求相匹配的文字或特定实例值。

[0050] 该实用工具提供了规定对在模板中出现的 EIT 和 EIV 的要求的能力。它提供了通过 Java 和 C# 中的类属所提供的能力的超集以及在 C++0X 中所提出的“概念”机制，而无需特殊的句法和相关联的限制。

[0051] 关于翻译器可扩展性的现有技术已提出了使内部表示具体化以允许扩展的指定（如用 LISP）与使用外部表示作为语法扩展的基础（如用 Xoc）之间的分歧。在本方法中，内部表示类型可嵌在外部表示（即，编程语言语法）中，从而实现两种现有技术方法中最好的方法。

[0052] 解释器结构

[0053] 在一个实施例中，解释器包括将文本解释输入翻译成内部所谓的字节代码的解析器，该字节代码由字节代码解释器执行。这种字节代码解释器方法采用了本领域公知的技术。它允许从输入的执行中模块化分离语言解析和错误检查以及有效地执行比输入执行更多次的语句，诸如迭代语句（诸如“for”循环）中所指定的那些语句。

[0054] 在这种解释器结构中，存在用于对变量、常规控制语句的通常操作和用于定义 IR 中的类型和数据元的字节代码。

[0055] 优选地，解释器还支持“监听器”对象的定义和例示，这些“监听器”对象响应于指

定对象变化来调用。这些指定对象包括 EIT 实例，这些 EIT 实例包括预定义 EIT。

[0056] DIM 操作

[0057] DIM 可操作如下：

[0058] a) 当遇到包括 EIV 的语言结构时，该结构被写入 EIT 对象，使得解释器中的程序被调用。

[0059] b) 解释器程序执行与该结构相关联的内部代码，从而可能重新调用 DIM、并且一旦新调用的 DIM 到达该输入的末尾就从该执行返回到调用 DIM 的执行。

[0060] 在此，解释器访问到目前为止所解析的程序的 IR（并且由此访问所有符号）。此外，符号表元素被有效地标记为文字或 EIT 变量，如之前所述。以下若干章节更详细地讨论了特定功能。

[0061] 优选地，解析器处理包含 EIV 而不是调用嵌入式解释器的特定简单结构。例如，使用诸如名称中的？和＊之类的通配符字符可由解析器处理，而不是调用嵌入式解释器来评估通配符以提供一系列的匹配。

[0062] 文件蕴含

[0063] 文件蕴含可通过语句来处理，该语句指定与翻译器输入流相对应的 EIV，例如，

[0064] translatorIn ＜＜″ filename″（文件名）；

[0065] 在此，translatorIn 是与翻译器输入流相对应的预定义 EIV。文件还可被指定为 EIV，并且随后必须被绑定到此时的特定常量串。它还可以是如上正常命名的常量串。

[0066] 响应于遇到以上语句，DIM 用该结构写入 EIT 对象，从而调用使解释器访问指定文件并用该输入文件调用 DIM 的新实例的对应解释器程序，随后等待这个新实例在恢复调用的 DIM 之前完成。

[0067] 条件编译 / 翻译

[0068] 条件编译可由解释器来处理，该解释器评估“if”语句的测试条件并且随后将“true（真）”部分中的令牌传递给 DIM。具体而言，DIM 解析“if”语句并且随后在确定测试表达式取决于 EIV 之后调用嵌入式解释器。解释器确定测试表达式的值并且随后读取输入的令牌，从而传递“true”部分中的那些令牌以供解析并且丢弃其余令牌。在一个实施例中，在这种情况下，DIM 有效地将“if cond then（如果条件，则）”推送到解释器堆栈上并且调用解释器。

[0069] 迭代生成

[0070] “for”循环可由解释器处理，该解释器执行 for 循环、例示指定数量的令牌实例、在贯穿 for 循环的每个迭代上用其定义来替代与循环变量相对应的 EIV 变量。在此，for 循环使用作为 EIV 的循环变量或者 EIT 类型的迭代器对象。

[0071] 模板定义和例示

[0072] 通过参数化的类型定义，解释器可调用 DIM 来定义内部表示中的模板对象。通过参数化的类型例示，解释器可被调用以用其定义来替代每个类型变量。

[0073] 嵌入式解释运算符

[0074] 在一个实施例中，基于对自变量类型的确定，每个运算符符号被视为指定嵌入式解释对象，该嵌入式解释对象基于可能只在编译时已知的这些自变量的类型和集体特性而绑定到指定过程。这可被视为常规运算符超载的扩展。例如，考虑具有自变量 a 和 b 的运

算符 f(a,b)。常规运算符超载基于在翻译期间根据如在翻译之前给定的 a 和 b 的类型来选择 f。在本方法中，可在翻译期间计算 a 和 / 或 b 的类型（即，这些类型本身可以是 EIV），并且随后可根据如在翻译期间所计算的 a 和 b 的类型来选择 f。

[0075] 扩展处理

[0076] 在一个实施例中，自动生成各种各样的对象的监听器类型以处理动态探针、纪录、远程访问和其他能力。每个实用工具或扩展被指定为由翻译器解释器执行的解释器程序。实际上，解释器程序在翻译期间监听事件并且被调用以采取行动，诸如响应于新的类型定义来定义对应的（嵌套的）类型。监听器还能够在内部对象模型上迭代，从而生成如特定实用工具所需的附属类型和过程。

[0077] 通常，需要单独的工具来提供具有附带开销和编程复杂性的每个这样的实用工具。在另一种常规方法中，每个这样的扩展被显式地编码到翻译器程序中。本方法的上述简单性与这些常规替换方法的复杂性形成鲜明对比。

[0078] 示例

[0079] 为了更好地理解本方法，考虑以下示例是有帮助的。

[0080] 假设网络交换机具有通过接口连接到有线链路的一些数量的（numPorts 个）端口，有线链路使该网络连接到网络的其余部分。一旦软件被编译，numPorts 的值就被固定，但在设计软件时可能是未知的。此外，在一些情况下，使对应于端口的变量名称静态地赋值为“port0”、“port1”等是合乎需要的。同样，通常存在与到控制交换机的 CPU（中央处理单元）的连接相对应的区别端口。此外，不同的交换机可具有不同类型的端口，诸如处理以太网协议的那些端口与处理 ATM 协议的那些端口。同样，在一些情况下，可能想将端口名称前缀“port”（端口）改变成另一个指定，例如“interface”（接口）。

[0081] 在本发明的一个实施例中，以上可被指定为：

[0082]

```
type Switch : Object {
   for(Interp::Int i=0;i<=numPorts;++i) {
      Interp::AttrName attrname = pnp+i;
      if(i==numPorts) {
         Interp::AttrName cpuname = "cpuport";
         CpuPort cpuname(attrname,i,this);
      }
      else PortType attrname(attrname,i,this);
   }
}
```

[0083] 在以上示例中，CpuPort 命名预定义类型，该预定义类型采用作为参数的串、整数和指向类型 Switch（交换机）的（反向）指针。PortType 指定类型 Interp::TypeName 的 EIV，该 EIV 在翻译中的该点处被绑定到特定类型。Interp::Int 和 Interp::AttrName 分别为整数类型和属性名称类型指定嵌入式解释器类型。numPorts 变量是类型 Interp::Int 的

预定义 EIV。绑定到文字串“port”的 pnp 变量是类型 Interp::String 的预定义 EIV。在该示例中，翻译器将诸如上述的输入翻译成如下的 C++ 类中间语言。

[0084] 当 DIM 遇到以上“for”语句时，由于其声明为类型 Interp::Int 即一个 EIT，它将循环变量“i”识别为 EIV。这种识别使它调用翻译器的嵌入式解释器模块，从而在此问题上提供初始结构的指示以及它在输入源中的位置。

[0085] 解释器识别“for”循环并且检查相关联的 EIV 可在此（翻译）时被评估。具体而言，它确定“i”可用其初始值来评估，因为初始值被指定为常量，即值 0。它还检查此时 numPorts 的值是已知的，从而以其他方式生成错误信息。numPorts 的值可通过在翻译器被调用时对翻译器的输入、通过类似于在诸如 Unix 和 Linux 之类的公共计算机操作系统中提供的所谓环境变量的机制、或者间接地通过其相关于另一个 EIV 的定义来指定。在该特定示例中，假设 numPorts 被设为值 4。此外，假设 PortType EIV 绑定到 EthPort 类型，该 EthPort 类型处理以太网协议。EIV pnp 绑定到串“port”。

[0086] 解释器随后继续有效地执行该“for”循环。语句

[0087] Interp::AttrName attrname = pnp+i ;

[0088] 被识别为定义从 EIV pnp 中形成的新属性名称，EIV pnp 具有与值 i 级联的文字值“port”。如本领域所公知的，整数值 i 被转换成为对应的文字串以匹配“+”运算符所需的类型，该“+”运算符此处基于该运算符的第一自变量 pnp 被解释为串级联运算符，该文字串为字符串。在该循环的初始翻译时执行时，值 i 为 0，由此 EIV attrname 的值为“port0”。

[0089] 下一个输入行被识别为其测试条件基于 i 和 numPorts 的已知值来评估的“if”语句，从而确定该测试条件是错误的，使它前进到“else”语句。

[0090] “else”输入行被解析为其名称由 EIV attrname 指定的属性的声明，具有参数 EIV attrname 和 i。由此，解释器生成对应的声明，从而用它们对应的值来替代 EIV。具体而言，循环的第一迭代上的这一行产生无 EIV 声明 :

[0091] EthPort port0(″ port0″ ,0, this) ;

[0092] 注意，作为参数的 attrname 值基于为串类型的端口类型的参数类型自动地被翻译成文字串。

[0093] 解释器随后识别循环的结束并且继续返回到“for”语句，从而使值 i 加 1、对照 numPorts 的值测试它、并且随后再次执行循环主体，这次值 i 为 1。循环主体的此后续执行使解释器生成无 EIV 声明 :

[0094] EthPort port1(″ port1″ ,1, this) ;

[0095] 这个过程重复，i 为 2 并且随后为 3。在循环的下一执行中，解释器识别 i 现在等于 numPorts、从而使它执行“if”语句，并且由此生成 EIV 声明 :

[0096] CpuPort cpuport(″ port4″ ,4, this) ;

[0097] 在循环的后续迭代中，解释器识别到值 i 不再小于 numPorts 并且终止循环的执行。它还识别“for”循环的结束对应于被调用来进行处理的语言结构的结束。因此，它重新调用 DIM 来处理它已生成的结果，该结果是有效的 :

[0098]

```
EthPort port0("port0",0,this);
EthPort port1("port1",1,this);
EthPort port2("port2",2,this);
EthPort port3("port3",3,this);
CpuPort cpuport("cpuport",4,this);
```

[0099] 在一个实施例中，DIM 生成各个输入类型作为 C++ 类以及各个属性声明作为那个类的数据成员。因此，DIM 生成 C++ 类类声明：

[0100]

```
class Switch : public Object {
public:
    EthPort * port0() const { return port0_; }
    ...
private:
    EthPort* port0_;
    EthPort* port1_;
    EthPort* port2_;
    EthPort* port3_;
```

[0101]

```
    CpuPort * cpuPort_;
};
```

[0102] 在此，遵循使数据成员与后缀为“_”字符的属性同名的惯例以区分任何对应的成员功能（诸如在类中声明的“port0()”成员函数）。沿着提供对私有数据成员“port0_”的访问的单个成员函数“port0”的行，″ …″ 所指定的部分指示可以是该类声明的一部分的附加声明。

[0103] 除了上述以外，DIM 生成用于交换机类的 C++ 结构函数，该 C++ 结构函数用对应类的实例来初始化这些数据成员，如下：

[0104]

```
Switch::Switch():
    port0_(new EthPort("port0",0,this),
    port1_(new EthPort("port1",1,this),
    port2_(new EthPort("port2",2,this),
    port3_(new EthPort("port3",3,this),
    cpuport_(new CpuPort("cpuport",4,this),
    {
    }
```

[0105] 如该示例中所示的，可基于只改变 numPorts 的定义，使所生成的软件适应翻译时所需的端口数量而无需改变输入源。它还可处理通过在翻译时改变绑定到 PortType 的值来简单地生成具有不同的端口类型的属性。端口名称前缀还可通过将 pnp 的定义改变成另一个串来改变，例如，

[0106] Interp::String pnp = " intf" ;

[0107] 因为 EIV 和 EIT 是输入语法的一部分，所以翻译器能够执行正常的句法检查并且提供信息错误检查。例如，如果因为程序员偶然在文字串“port”的开头插入“.”所以输入源指定

[0108] Interp::String pnp = " .port" ;

[0109] 并且如果不允许该字符作为合法属性名称的一部分，则嵌入式解释器可在它遇到以下时发出特别指示所生成的“port”名称不是合法属性名称的出错信息：

[0110] Interp::AttrName attrname = pnp+i ;该示例还示出了本方法可减小需要被指定的输入源的量，即使当 numPorts 不变时。相应地，它允许程序员在输入源中指示其扩展形式不那么明显的重复结构中的规律性。此外，如果属性被指示为要静态地定义，则扩展版本还可通过生成扩展静态形式来允许降低执行时的开销，从而消除了对其运行时生成的需要。

[0111] 最后，以上示例可被变换成模板，从而通过用 EIV 参数限定声明来允许具有指定参数的延迟例示。例如，该示例可被模板化为：

[0112]

```
type SwitchTemplate(Interp::Port PortType,Interp::Int numPorts, Interp::String
pnp) : Object {
    for(Interp::Int i=0;i<=numPorts;++i) {
        Interp::AttrName attrname = pnp+i;
        if(i==numPorts) {
            Interp::AttrName cpuname = "cpuport";
            CpuPort cpuname(attrname,i,this);
        }
        else PortType attrname(attrname,i,this);
    }
}
```

[0113] “SwitchTemplate”之后的括号内的EIV PortType、numPorts和pnp指示这是具有这些EIV作为参数的参数化类型声明。由此，当DIM遇到该类型声明时，这些EIV声明的出现使它调用解释器，该解释器识别还未绑定的这些EIV声明并且延迟例示。具有特定参数的这种类型的后续声明使特定类型被例示。例如，声明：

[0114] Interp::Type pt = isEtherSwitch ? EthPort:AtmPort ;

[0115] Interp::Int ports = 3 ;

[0116] SwitchTemplate(pt,ports, " port")switch ;

[0117] 可使类型的例示与在该示例的开始所指定的交换机类型相当，假设EIV isEtherSwitch为真、数据元称为该类型的“switch”。可与上述一样来实现这种例示的交换机类型，不同之处在于作为该例示的一部分的三个EIV可绑定到调用它的自变量“pt”、“ports”和“port”。

[0118] 与在思考之后被添加到语言的宏处理器或C++模板实用工具所提供的替换能力不同，这种能力使用语言的现有句法，从而依赖于EIT的类型标签化来区分翻译时评估和运行时评估。此外，作为语言中的正确类型，可传递诸如“port”之类的串以及表达式并具有准确的类型检查，并且可施加转换。例如，上述可被重新写入以传递表达式，从而如下地消除EIV pt：

[0119] SwitchTemplate(isEtherSwitch ? EthPort :AtmPort, ports, " port ") switch ;

[0120] 假设“isEtherSwitch”是一EIV。

[0121] 如该示例示出地，在本方法中，通过使用嵌入式解释类型和它们相关联的变量、调用解释器以在翻译时执行这些细节，编程语言的正常语法和强大结构可在翻译时用来修改、限定和扩展所生成的软件。

[0122] 前面的描述集中在根据本发明实施例的方法上。本发明的实施例还包括被编程以自动地执行相关方法的计算机装置（即，系统）和包含执行相关方法的软件指令的计算机可读介质。这些计算机装置可用硬件和/或软件的任意组合来提供。计算机可读介质包括

诸如磁和光介质之类的大容量存储介质，并且可以是本地的（即，直接连接到计算机）或远程的（即，通过一个或多个网络连接到计算机）。