

[19] 中华人民共和国国家知识产权局



[12] 发明专利申请公开说明书

[21] 申请号 200510126754.2

[51] Int. Cl.

G06F 9/45 (2006.01)

G06F 9/44 (2006.01)

[43] 公开日 2006 年 5 月 24 日

[11] 公开号 CN 1776621A

[22] 申请日 2005.11.18

[21] 申请号 200510126754.2

[30] 优先权

[32] 2004.11.19 [33] JP [31] 336539/2004

[71] 申请人 松下电器产业株式会社

地址 日本大阪府

[72] 发明人 田中裕久

[74] 专利代理机构 永新专利商标代理有限公司

代理人 胡建新

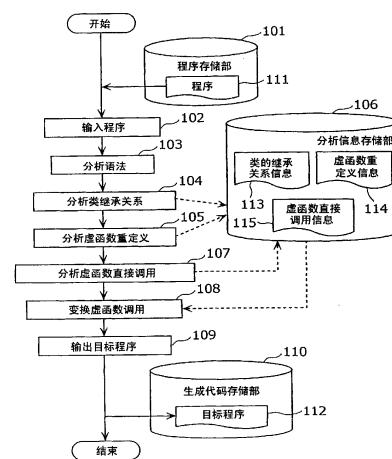
权利要求书 5 页 说明书 41 页 附图 48 页

[54] 发明名称

程序变换方法

[57] 摘要

本发明的程序变换方法，着眼于有时能够确定成员函数的 this 指针的类型的事实，能够提高执行性能，减小代码长度，将虚函数调用变换为函数的直接调用，它变换用面向对象语言描述的程序，包含：方法分析步骤，在上述程序中，分析调用方法的实例的类型；虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法的调用；以及虚方法调用变换步骤，根据上述方法分析步骤的分析结果，将上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。



1. 一种程序变换方法，变换用面向对象语言描述的程序，其特征在于，包含：

方法分析步骤，在上述程序中，分析调用方法的实例的类型；

虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；以及

虚方法调用变换步骤，根据上述方法分析步骤的分析结果，将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

2. 如权利要求1所述的程序变换方法，其特征在于，

上述方法分析步骤包含：

类继承关系分析步骤，在上述程序中，分析类的继承关系；

虚方法重定义分析步骤，分析上述类中的上述虚方法的重定义状况；

虚方法直接调用分析步骤，在上述程序中，分析上述虚方法的直接调用；以及

实例确定步骤，根据上述类继承关系分析步骤、上述虚方法重定义分析步骤及上述虚方法直接调用分析步骤的分析结果，来确定调用上述方法的上述实例的类型；

在上述方法分析步骤能够确定上述实例的类型的情况下，上述虚方法调用变换步骤将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

3. 如权利要求2所述的程序变换方法，其特征在于，上述实例确定步骤包含下述步骤：

判定上述方法所属的类是否具有派生类；

在上述类具有派生类的情况下，判定是否在直接继承上述类的所有类中重定义了上述方法；以及

判定上述类的所有派生类的实例是否直接调用了上述方法。

4. 如权利要求2所述的程序变换方法，其特征在于，上述实例确定步骤包含下述步骤：

判定上述方法所属的类是否具有派生类；

在上述类具有派生类的情况下，判定是否在对由上述虚方法调用提取步骤提取出的上述虚方法进行重定义的第二类中重定义了上述方法；以及

判定上述第二类及上述第二类的所有派生类的实例是否直接调用了上述方法。

5. 如权利要求1所述的程序变换方法，其特征在于，

在上述方法是实例生成方法时，上述方法分析步骤确定调用上述方法的上述实例的类型。

6. 如权利要求1所述的程序变换方法，其特征在于，

在上述方法是实例删除方法时，上述方法分析步骤确定调用上述方法的上述实例的类型。

7. 一种程序变换方法，变换用面向对象语言描述的程序，其特征在于，包含：

方法分析步骤，在上述程序中，分析调用方法的实例的类型；

虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；

虚方法复制步骤，根据上述方法分析步骤的结果，复制上述虚方法；以及

虚方法调用变换步骤，根据上述方法分析步骤的分析结果，将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

8. 如权利要求7所述的程序变换方法，其特征在于，

上述方法分析步骤包含：

类继承关系分析步骤，在上述程序中，分析类的继承关系；

虚方法重定义分析步骤，分析上述类中的上述虚方法的重定义状况；

虚方法直接调用分析步骤，在上述程序中，分析上述虚方法的直接

调用；以及

实例确定步骤，根据上述类继承关系分析步骤、上述虚方法重定义分析步骤及上述虚方法直接调用分析步骤的分析结果，确定调用上述方法的上述实例的类型；

在上述方法分析步骤能够确定上述实例的类型的情况下，上述虚方法调用变换步骤将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

9. 如权利要求8所述的程序变换方法，其特征在于，

上述实例确定步骤包含下述步骤：

判定上述方法所属的类是否具有派生类；和

判定上述类的所有派生类的实例是否直接调用了上述方法；

在上述虚方法复制步骤中，对上述程序中包含的所有类，在上述类具有基类的情况下，将上述基类中包含的虚方法中的未在上述类中重定义的上述虚方法，在上述类中进行重定义。

10. 如权利要求8所述的程序变换方法，其特征在于，

上述实例确定步骤包含下述步骤：

判定上述方法所属的类是否具有派生类；和

判定上述类的所有派生类的实例是否直接调用了上述方法；

在上述虚方法复制步骤中，对直接继承上述方法所属的类的所有第二类，当在上述第二类中未重定义上述方法时，在上述第二类中重定义上述方法。

11. 如权利要求8所述的程序变换方法，其特征在于，

上述实例确定步骤包含下述步骤：

判定上述方法所属的类是否具有派生类；

在上述类具有派生类的情况下，提取对由上述虚方法调用提取步骤提取出的上述虚方法进行重定义的第三类；以及

判定上述第三类及上述第三类的所有派生类的实例是否直接调用了上述方法；

在上述虚方法复制步骤中，当在上述第三类中未重定义上述方法时，在上述第三类中重定义上述方法。

12. 一种程序变换方法，变换用面向对象语言描述的程序，其特征在于，

包含：

编译步骤，编译上述程序；和

结合步骤，结合上述编译步骤的编译结果；

上述编译步骤包含：

方法分析步骤，在上述程序中，分析调用方法的实例的类型；

虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；以及

输出步骤，输出上述方法分析步骤的分析结果、上述虚方法调用提取步骤的分析结果及上述程序的变换结果；

上述结合步骤包含：

分析数据提取步骤，输入上述编译步骤的输出，提取上述编译步骤的输出中包含的分析结果；

虚方法调用变换步骤，根据上述分析结果提取步骤的结果，将上述虚方法调用变换为直接方法调用；以及

链接步骤，链接上述虚方法调用变换步骤的结果。

13. 一种程序变换方法，变换用面向对象语言描述的程序，其特征在于，

包含：

预编译步骤，从上述程序中提取规定的信息；和

变换步骤，根据上述预编译步骤的提取结果，将虚方法调用变换为直接方法调用；

上述预编译步骤包含：

方法分析步骤，在上述程序中，分析调用方法的实例的类型；和

虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方

法调用；

上述变换步骤包含：

输入步骤，接收上述程序、上述方法分析步骤的分析结果及上述虚方法调用提取步骤的分析结果，作为输入；和

虚方法调用变换步骤，根据由上述输入步骤输入的分析结果，将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

程序变换方法

技术领域

本发明涉及变换用面向对象语言描述的程序的程序变换方法，特别涉及将虚函数调用变换为直接函数调用的程序变换方法。

背景技术

近年来，随着程序的大规模化，为了提高程序的开发效率、实现再利用，或者为了提高程序的可维护性，频繁用 C++ 或 Java(注册商标)等面向对象语言进行程序开发。

作为面向对象语言的特征之一，有多态性。所谓“多态性”，表示其动作按照执行程序时的对象实例的类型来动态地变化的性质。

为了实现该多态性，例如在 C++ 语言中具有称作虚函数的功能。用图 1~图 4 来说明该虚函数。

图 1 是用面向对象语言 C++ 描述的程序。类 A 的成员函数 f、g 及 h 的声明中指定的关键词“virtual”指定了成员函数 f、g 及 h 是虚函数。所谓“虚函数”，是指在执行程序时按照指针指向的对象的类型来动态决定要执行的函数的函数，在派生类中重定义了基类的虚函数的情况下，在指针指向派生类的对象的情况下，执行派生类的虚函数。

类 B 继承了类 A，类 B 的成员函数 g 覆盖了基类 A 的虚函数 g，执行类 B 特有的动作。

此时，在函数 t2 内指针变量 p 调用成员函数 g 后，成为虚函数调用。此时，如果指针变量 p 指向的对象是类 A 的对象，则执行类 A 的函数 g；如果是类 B 的对象，则执行类 B 的函数 g。这样，按照指针变量 p 指向的对象的类型在执行时决定要执行的函数。

该虚函数的实现方法的概略图示于图 2。

类 B 类型的对象 obj_b 在内部保持指向虚函数表的指针成员 vptr。成员 vptr 指向类 B 的虚函数表 vtbl_B。在类 B 的虚函数表中，保持着可由类 B 调用的所有虚函数的地址。

考虑指针变量 p 调用虚函数 g 的情况。

在指针变量 p 指向类 B 类型的对象 obj_b 的情况下，p 保持着 obj_b 的起始地址。虚函数调用所调用的函数的地址需要通过下述过程来取得：首先取得虚函数表，接着取得要调用的函数的地址。

即，取得将指针变量 p 的值加上 offset1 所得的地址的内容，该值加上 offset2 所得的地址的内容成为要调用的函数的地址。

因此，执行虚函数调用的汇编码如图 3 所示。而执行函数的直接调用的汇编码如图 4 所示，虚函数调用的代码往往指令数多，执行速度也慢。

因此，尽量减少虚函数调用，程序的执行性能将提高。

作为用于减少执行虚函数调用的次数、提高执行性能的现有技术，有下述技术：比较虚函数调用所调用的函数的地址，在要调用的函数是某个特定的函数的情况下直接调用该函数(例如参照(日本)特开 2000-40007 号公报)。由此，可以变换为在能够确定虚函数调用所执行的函数的情况下执行直接调用该函数的代码，而在不能确定的情况下执行通常的虚函数调用的代码。用特开 2000-40007 号公报记载的方法变换了图 5 所示的程序的箭头(A)的函数 h 的概略图示于图 6。这样，在虚函数的地址是成员函数“A::f”(由类 A 定义的函数 f)的地址的情况下，直接调用函数“A::f”；而在其他情况下，进行虚函数调用。进而，可以通过内联展开函数的直接调用来提高执行性能。

此外，有下述技术：分析类的继承关系，将没有派生类的类的虚函数调用变换为函数的直接调用(例如参照(日本)特开平 10-320204 号公报)。由此，可以在编译程序时将虚函数调用变换为函数的直接调用，所以能够提高程序的执行性能。用特开平 10-320204 号公报记载的方法变换

了图 5 所示的程序的箭头(A)的函数 h 的概略图示于图 7。

这里，在能够确定要调用虚函数的类的情况下，能够确定虚函数调用所调用的函数。例如，有下述情况：对于图 5 的箭头(B)所示的类 A 的成员函数 f，可以确定 this 指针的类型只能是类 A。此时，在 this 指针调用虚函数 g 的情况下执行的函数必定为类 A 的成员函数 g。这里，所谓 this 指针，表示在成员函数的内部指向启动了该成员函数的对象的特别的指针。

但是，在特开 2000-40007 号公报记载的技术中，如图 8 所示，即使在能够确定调用虚函数的类的情况下也生成比较函数的地址的代码，所以与直接调用函数的代码相比，有执行性能及代码长度变差的问题。

此外，由于 this 指针的类型是类 A，所以在特开平 10-320204 号公报记载的技术中，在调用虚函数的类具有派生类的情况下，有不能将虚函数调用变换为函数的直接调用的问题。

发明内容

本发明就是为了解决上述课题而提出的，其目的在于提供一种程序变换方法，着眼于有时能够确定成员函数的 this 指针的类型的事实，能够提高执行性能，而且减小代码长度，同时将虚函数调用变换为函数的直接调用。

为了解决上述现有的课题，本发明的程序变换方法，变换用面向对象语言描述的程序，其特征在于，包含：方法分析步骤，在上述程序中，分析调用方法的实例的类型；虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；以及虚方法调用变换步骤，根据上述方法分析步骤的分析结果，将上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

能够将虚方法(即虚函数)调用变换为直接方法调用(即函数的直接调用)，能够提高程序的执行性能。此外，能够减小程序的代码长度。

此外，本发明另一方面的程序变换方法，变换用面向对象语言描述

的程序，其特征在于，包含：方法分析步骤，在上述程序中，分析调用方法的实例的类型；虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；虚方法复制步骤，根据上述方法分析步骤的结果，复制上述虚方法；以及虚方法调用变换步骤，根据上述方法分析步骤的分析结果，将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

能够通过复制虚方法，将虚方法调用变换为直接方法调用的可能性提高。因此，能够进一步提高程序的执行性能，而且能够减小程序的代码长度。

此外，本发明另一方面的程序变换方法变换用面向对象语言描述的程序，其特征在于，包含：编译步骤，编译上述程序；和结合步骤，结合上述编译步骤的编译结果；上述编译步骤包含：方法分析步骤，在上述程序中，分析调用方法的实例的类型；虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；以及输出步骤，输出上述方法分析步骤的分析结果、上述虚方法调用提取步骤的分析结果及上述程序的变换结果；上述结合步骤包含：分析数据提取步骤，输入上述编译步骤的输出，提取上述编译步骤的输出中包含的分析结果；虚方法调用变换步骤，根据上述分析结果提取步骤的结果，将上述虚方法调用变换为直接方法调用；以及链接步骤，链接上述虚方法调用变换步骤的结果。

能够用与上述程序变换方法不同的结构，得到同样的程序执行性能。

此外，本发明另一方面的程序变换方法，变换用面向对象语言描述的程序，其特征在于，包含：预编译步骤，从上述程序中提取规定的信息；和变换步骤，根据上述预编译步骤的提取结果，将虚方法调用变换为直接方法调用；上述预编译步骤包含：方法分析步骤，在上述程序中，分析调用方法的实例的类型；和虚方法调用提取步骤，在上述方法的定义中，提取上述实例的虚方法调用；上述变换步骤包含：输入步骤，接收上述程序、上述方法分析步骤的分析结果及上述虚方法调用提取步骤

的分析结果，作为输入；和虚方法调用变换步骤，根据由上述输入步骤输入的分析结果，将由上述虚方法调用提取步骤提取出的虚方法调用变换为直接方法调用。

能够用与上述程序变换方法也不同的结构，得到同样的程序执行性能。

根据本发明，能够提供一种程序变换方法，能够提高执行性能，而且减小代码长度，同时将虚函数调用变换为函数的直接调用。

附图说明

通过以下结合示出了本发明的特定具体实施例的附图进行的描述，本发明的这些和其他目的、优点以及特性将变得很明显。在附图中：

图 1 是包含虚函数调用的程序例的图。

图 2 是虚函数调用机构的安装例的图。

图 3 是与虚函数调用对应的汇编码的图。

图 4 是与函数的直接调用对应的汇编码的图。

图 5 是用于说明现有技术的包含虚函数调用的程序例的图。

图 6 是特开 2000-40007 号公报记载的技术的变换结果的图。

图 7 是特开平 10-320204 号公报记载的技术的变换结果的图。

图 8 是现有技术中的课题的图。

图 9 是说明本发明实施方式 1 的程序变换装置执行的处理的概略的图。

图 10 是本发明实施方式 1 的程序变换装置的结构的图。

图 11 是本发明实施方式 1 的程序变换步骤的结构图。

图 12 是类继承关系分析处理(S104)的流程图。

图 13 是本发明实施方式 1 的程序存储部 101 中保存着的程序例的图。

图 14 是本发明实施方式 1 的类的继承关系信息 113 的图。

图 15 是虚函数重定义分析处理(S105)的流程图。

图 16 是本发明实施方式 1 的虚函数重定义信息 114 的图。

- 图 17 是虚函数直接调用分析处理(S107)的流程图。
- 图 18 是本发明实施方式 1 的虚函数直接调用信息 115 的图。
- 图 19 是虚函数调用变换处理(S108)的流程图。
- 图 20 是本发明实施方式 1 的虚函数调用变换步骤中生成的程序的图。
- 图 21 是说明本发明实施方式 2 的程序变换装置执行的处理的概略的图。
- 图 22 是本发明实施方式 2 的类的继承关系信息 113 的图。
- 图 23 是本发明实施方式 2 的虚函数重定义信息 114 的图。
- 图 24 是本发明实施方式 2 的虚函数直接调用信息 115 的图。
- 图 25 是本发明实施方式 2 的虚函数调用变换处理(S108)的流程图。
- 图 26 是本发明实施方式 2 的虚函数调用变换处理中生成的程序的图。
- 图 27 是说明本发明实施方式 3 的程序变换装置执行的处理的概略的图。
- 图 28 是本发明实施方式 3 的程序变换装置的结构的图。
- 图 29 是本发明实施方式 3 的程序变换步骤的结构图。
- 图 30 是本发明实施方式 3 的程序存储部 101 中保存着的程序例的图。
- 图 31 是本发明实施方式 3 的类的继承关系信息 113 的图。
- 图 32 是本发明实施方式 3 的虚函数重定义信息 114 的图。
- 图 33 是本发明实施方式 3 的虚函数直接调用信息 115 的图。
- 图 34 是虚函数调用变换判定处理(S201)的流程图。
- 图 35 是本发明实施方式 3 的可直接调用语句信息 116 的图。
- 图 36 是虚函数定义复制处理(S202)的流程图。
- 图 37 是本发明实施方式 3 的虚函数定义复制步骤中生成的类定义的图。
- 图 38 是本发明实施方式 3 的虚函数定义复制步骤中生成的虚函数表的图。

图 39 是虚函数调用变换处理(S203)的流程图。

图 40 是本发明实施方式 3 的虚函数调用变换步骤中生成的程序的图。

图 41 是说明本发明实施方式 4 的程序变换装置执行的处理的概略的图。

图 42 是本发明实施方式 4 的类的继承关系信息 113 的图。

图 43 是本发明实施方式 4 的虚函数重定义信息 114 的图。

图 44 是本发明实施方式 4 的虚函数直接调用信息 115 的图。

图 45 是本发明实施方式 4 的虚函数调用变换判定处理(S201)的流程图。

图 46 是本发明实施方式 4 的可直接调用语句信息 116 的图。

图 47 是本发明实施方式 4 的复制虚函数信息的图。

图 48 是本发明实施方式 4 的虚函数定义复制信息(S202)的流程图。

图 49 是本发明实施方式 4 的虚函数定义复制处理中生成的类定义的图。

图 50 是本发明实施方式 4 的虚函数定义复制处理中生成的虚函数表的图。

图 51 是本发明实施方式 4 的虚函数调用变换处理中生成的程序的图。

图 52 是说明本发明实施方式 5 的程序变换装置执行的处理的概略的图。

图 53 是本发明实施方式 5 的类的继承关系信息 113 的图。

图 54 是本发明实施方式 5 的虚函数重定义信息 114 的图。

图 55 是本发明实施方式 5 的虚函数直接调用信息 115 的图。

图 56 是本发明实施方式 5 的虚函数调用变换判定处理(S201)的流程图。

图 57 是本发明实施方式 5 的可直接调用语句信息 116 的图。

图 58 是本发明实施方式 5 的复制虚函数信息中保存着的信息的图。

图 59 是本发明实施方式 5 的虚函数定义复制步骤中生成的类定义的图。

图 60 是本发明实施方式 5 的虚函数定义复制步骤中生成的虚函数表的图。

图 61 是本发明实施方式 5 的虚函数调用变换步骤中生成的程序的图。

图 62 是本发明实施方式 6 的程序变换装置的结构的框图。

图 63 是本发明实施方式 6 的程序变换装置执行的处理的流程图。

图 64 是本发明实施方式 7 的程序变换装置的结构的框图。

图 65 是本发明实施方式 7 的程序变换装置执行的处理的流程图。

具体实施方式

[实施方式 1]

以下，参照附图来说明本发明的程序变换装置的实施方式的一例。

在本实施方式中，在满足以下的条件(1)至条件(4)的情况下，能够确定 this 指针的类型，所以将虚函数 F1 的调用变换为直接函数调用。

即，

条件(1)：在虚函数 F2 的定义中描述了虚函数 F1。

条件(2)：this 指针调用了虚函数 F1。

条件(3)：虚函数 F2 所属的类 C1 没有派生类，或者在类 C1 的所有直接派生类中重定义了虚函数 F2。

条件(4)：类 C1 的派生类的对象不直接调用虚函数 F2。

图 9 是说明实施方式 1 的程序变换装置根据上述 4 个条件执行的处理的概略的图。即，考虑给出了图 9(a)所示的类的定义、而且给出了图 9(b)所示的程序的情况。

在设虚函数 g 为虚函数 F1、设虚函数 f 为虚函数 F2 的情况下，图 9(b)的语句 “this->g()” 满足条件(1)及条件(2)。即，在虚函数 F2(虚函数 f)的定义中描述了虚函数 F1(虚函数 g)，所以满足条件(1)。此外，this 指

针调用了虚函数 F1(虚函数 g)，所以满足条件(2)。

此外，考虑条件(3)。如图 9(b)所示，虚函数 F2(虚函数 f)所属的类 C1 是类 A。因此，如图 9(a)所示，在类 C1(类 A)的所有直接派生类(即类 B)中重定义了虚函数 F2(虚函数 f)。因此，满足条件(3)。

再者，考虑条件(4)。在程序中不存在 A::f(void)的直接调用。因此，类 C1(类 A)的派生类(类 B、C、D)的对象不直接调用虚函数 F2(虚函数 f)。因此，满足条件(4)。

因此，在这种情况下，能够将图 9(b)所示的虚函数 g 的调用如图 9(c)所示变换为类 A 的虚函数 g 的直接调用。

[程序变换装置的结构]

图 10 是本发明实施方式 1 的程序变换装置 100 的结构的图。程序变换装置 100 是接受用面向对象语言——C++语言描述的程序、输出目标程序的装置，包括程序存储部 101、输入部 102、语法分析部 103、类继承关系分析部 104、虚函数重定义分析部 105、分析信息存储部 106、虚函数直接调用分析部 107、虚函数调用变换部 108、输出部 109、以及生成代码存储部 110。

程序存储部 101 是存储程序变换装置 100 中作为变换对象的、用面向对象语言——C++语言描述的程序 111 的存储装置。在程序存储部 101 中，存储着 1 个以上的程序 111。

输入部 102 是依次输入程序存储部 101 中保存着的所有程序 111、并输出到语法分析部 103 的处理部。

语法分析部 103 是分析从输入部 102 接收到的程序 111 的语法、并进行符号表的生成或语法树的生成等、将其分析结果交给类继承关系分析部 104 的处理部。

类继承关系分析部 104 是提取程序 111 中包含的所有类并分析类间的继承关系、生成分析结果——类的继承关系信息的处理部。

虚函数重定义分析部 105 是分析类的虚函数是否重定义了基类的虚函数、生成分析结果——虚函数重定义信息的处理部。

虚函数直接调用分析部 107 是分析程序中直接调用的虚函数及启动对象、并生成虚函数直接调用信息的处理部。

分析信息存储部 106 是存储类继承关系分析部 104 生成的类的继承关系信息 113、虚函数重定义分析部 105 生成的虚函数重定义信息 114 及虚函数直接调用分析部 107 生成的虚函数直接调用信息 115 的存储装置。

虚函数调用变换部 108 是参照分析信息存储部 106 中保存着的类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115、判定能否将虚函数调用变换为函数的直接调用、并在能够变换的情况下变换为函数的直接调用的处理部。

输出部 109 是将虚函数调用变换部 108 的变换结果作为目标程序 112 保存到生成代码存储部 110 中的处理部。

生成代码存储部 110 是存储目标程序 112 的存储装置。

[程序变换装置执行的处理的概要]

图 11 是本发明实施方式 1 的程序变换装置 100 执行的处理的流程图。

输入部 102 依次输入程序存储部 101 中保存着的所有程序 111，并交给语法分析部 103(S102)。

语法分析部 103 分析从输入部 102 接收到的程序 111 的语法，进行符号表的生成和语法树的生成等，并将其分析结果交给类继承关系分析部 104(S103)。

类继承关系分析部 104 提取程序 111 中包含的所有类并分析类间的继承关系，将分析结果作为类的继承关系信息 113 保存到分析信息存储部 106 中(S104)。

虚函数重定义分析部 105 分析类的虚函数是否重定义了基类的虚函数，将分析结果作为虚函数重定义信息 114 保存到分析信息存储部 106 中(S105)。

虚函数直接调用分析部 107 分析程序 111 中直接调用的虚函数及启动对象，将分析结果作为虚函数直接调用信息 115 保存到分析信息存储部 106(S107)。

虚函数调用变换部 108 参照分析信息存储部 106 中保存着的类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115，判定能否将虚函数调用变换为函数的直接调用，在能够变换的情况下变换为函数的直接调用(S108)。

输出部 109 将虚函数调用变换部 108 的变换结果作为目标程序 112 保存到生成代码存储部 110 中(S109)。

其中，输入步骤 S102、语法分析步骤 S103、输出步骤 S109 不是本发明的重点，所以省略详细说明。

[程序变换装置执行的处理的细节]

接着，参照程序例来说明程序变换装置 100 执行的处理的细节。

首先，输入部 102 从程序存储部 101 输入所有程序 111(S102)，语法分析部 103 进行程序 111 的语法树和符号表的生成等(S103)。

接着，说明类继承关系分析处理(S104)的细节。图 12 是类继承关系分析处理(S104)的详细流程图。这里，作为输入到程序变换装置 100 中的程序 111，考虑图 13 所示的程序 111。

类继承关系分析处理(S104)包含从 a1 到 a7 的步骤。以下按记号的顺序来进行说明。

在步骤 a1 中，类继承关系分析部 104 对语法分析部 103 分析出的所有程序 111 中包含的所有类 C1 重复以下的处理。即，对图 13 的所有程序 111 中包含的类 A、B、C 及 D 重复以下的处理。

在步骤 a2 中，类继承关系分析部 104 检查类 C1 是否已经记录在类的继承关系信息 113 中。在已经记录的情况下，将处理转移到步骤 a7；而在未记录的情况下，将处理转移到步骤 a3。在一开始分析图 13 的类 A 时步骤 a2 的判定为“否”，将处理转移到步骤 a3。而在其后分析类 A 时步骤 a2 的判定为“是”，将处理转移到步骤 a7。对类 B、C 及 D 也同样。

在步骤 a3 中，类继承关系分析部 104 将类 C1 登录到类的继承关系信息 113 中。在适用于图 13 的类 A 的情况下，向类的继承关系信息 113 中登录类 A。对类 B、C 及 D 也同样。

在步骤 a4 中，类继承关系分析部 104 检查类 C1 是否具有基类 C2。在类 C1 没有基类 C2 的情况下，将处理转移到步骤 a5；而在具有基类 C2 的情况下，将处理转移到步骤 a6。在图 13 的类 A 的情况下，类 A 没有基类。因此步骤 a4 的判定为“否”，将处理转移到步骤 a5。而在类 B 的情况下作为基类而具有类 A，所以步骤 a4 的判定为“是”，将处理转移到步骤 a6。类 C 及 D 也分别具有基类 B，所以步骤 a4 的判定为“是”，将处理转移到步骤 a6。

在步骤 a5 中，类继承关系分析部 104 生成类 C1 的类树，登录到类的继承关系信息 113 中(a5)。所谓类树，是指表示类的继承关系的图。在图 13 的类 A 的情况下，类继承关系分析部 104 对类 A 生成类树，向类的继承关系信息 113 中登录生成的类树。

在步骤 a6 中，类继承关系分析部 104 将类 C1 作为类 C2 的派生类添加到 C2 所属的类树中，更新类树。在图 13 的类 B 的情况下，将类 B 作为类 A 的派生类登录到类 A 所属的类树中。将类 C 及类 D 也同样作为类 B 的派生类登录到类树中。

在步骤 a7 中，将处理转移到步骤 a1，重复循环的处理。

图 14 是类继承关系分析部 104 分析图 13 的程序 111 的结果的类的继承关系信息 113 的图。在类的继承关系信息 113 中，示出了类 B 是类 A 的派生类，类 C 及 D 是类 B 的派生类。

接着，说明虚函数重定义分析处理(S105)的细节。图 15 是虚函数重定义分析处理(S105)的详细流程图。这里，作为输入到程序变换装置 100 中的程序 111，考虑图 13 所示的程序 111。

虚函数重定义分析处理(S105)包含从 b1 到 b8 的步骤。以下按记号的顺序来进行说明。

在步骤 b1 中，虚函数重定义分析部 105 对类的继承关系信息 113 中包含的所有类 C1 重复以下的处理。即，对图 14 所示的类 A、B、C 及 D 的各个重复以下的处理。

在步骤 b2 中，虚函数重定义分析部 105 将类 C1 的所有虚函数的重

定义标记设为 OFF(“关”), 记录到虚函数重定义信息 114 中。在类 A 的情况下, 对虚函数 f、g 及 h 将重定义标记设为 OFF, 记录到虚函数重定义信息 114 中。对类 B、C 及 D 也同样进行记录。

在步骤 b3 中, 虚函数重定义分析部 105 检查类 C1 是否具有基类 C2。在类 C1 具有基类 C2 的情况下, 将处理转移到步骤 b4; 而在没有基类 C2 的情况下, 将处理转移到步骤 b8。在图 14 的类的继承关系信息 113 所示的类 A 的情况下, 类 A 没有基类, 所以步骤 b3 的判定为“否”, 将处理转移到步骤 b8。而在类 B 的情况下作为基类而具有类 A, 所以步骤 b3 的判定为“是”, 将处理转移到步骤 b4。类 C 及 D 也分别具有基类 B, 所以步骤 b3 的判定为“是”, 将处理转移到步骤 b4。

在步骤 b4 中, 虚函数重定义分析部 105 对类 C1 的所有虚函数 F1 重复以下的处理。在图 13 的类 B 的情况下, 对类 B 的虚函数 f、g 及 h 重复以下的处理。

在步骤 b5 中, 虚函数重定义分析部 105 检查虚函数 F1 是否覆盖了类 C1 的基类 C2 的虚函数。在虚函数 F1 覆盖了基类 C2 的虚函数的情况下将处理转移到步骤 b6, 而在未覆盖的情况下将处理转移到步骤 b7。在图 13 的类 B 的虚函数 f 的情况下, 覆盖了基类 A 的虚函数 f, 所以步骤 b5 的判定为“是”, 将处理转移到步骤 b6。而在类 B 的虚函数 h 的情况下, 未覆盖基类 A 的虚函数 h, 所以步骤 b5 的判定为“否”, 将处理转移到步骤 b7。

在步骤 b6 中, 虚函数重定义分析部 105 将虚函数 F1 的重定义标记设为 ON(“开”)。在图 13 的类 B 的虚函数 f 的情况下, 将与类 B 的虚函数 f 对应的重定义标记设定为 ON。

在步骤 b7 中, 虚函数重定义分析部 105 将处理转移到步骤 b4, 重复循环的处理。

在步骤 b8 中, 虚函数重定义分析部 105 将处理转移到步骤 b1, 重复循环的处理。

图 16 是示出用虚函数重定义分析部 105 分析图 13 的程序 111 的结

果的虚函数重定义信息 114 的图。虚函数重定义信息 114 的左栏示出了类名，中栏示出了虚函数名，右栏示出了虚函数是否重定义了。在 ON 的情况下表示重定义了，而在 OFF 的情况下表示未重定义。例如，在类 B 中，示出了虚函数 f 和 g 重定义了，而虚函数 h 未重定义。

接着，说明虚函数直接调用分析处理(S107)的细节。图 17 是虚函数直接调用分析处理(S107)的详细流程图。这里，作为输入到程序变换装置 100 中的程序 111，考虑图 13 所示的程序 111。

虚函数直接调用分析处理(S107)包含从 c1 到 c4 的步骤。以下按记号的顺序来进行说明。

在步骤 c1 中，虚函数直接调用分析部 107 对语法分析处理(S103)分析出的所有程序 111 中的所有语句 S1 重复以下的处理。在图 13 中，“this->g();” 和 “obj_c.g();” 等成为处理的对象。

在步骤 c2 中，虚函数直接调用分析部 107 检查在语句 S1 中是否包含虚函数 F1 的直接调用。在语句 S1 中包含虚函数 F1 的直接调用的情况下将处理转移到步骤 c3，而在不包含直接调用的情况下将处理转移到步骤 c4。在图 13 的语句 “this->g();” 中，不直接调用类 A 的虚函数 g，所以步骤 c2 的判定为“否”，将处理转移到步骤 c4。而在语句 “obj_c.g();” 中，直接调用了类 B 的虚函数 g，所以步骤 c2 的判定为“是”，将处理转移到步骤 c3。

在步骤 c3 中，虚函数直接调用分析部 107 将虚函数 F1 的启动对象的类 C1 和虚函数 F1 记录到虚函数直接调用信息 115 中。在图 13 的语句 “obj_c.g();” 中，将启动对象的类 C、及 “B::g” 记录到虚函数直接调用信息 115 中。

在步骤 c4 中，虚函数直接调用分析部 107 将处理转移到步骤 c1，重复循环的处理。

图 18 是虚函数直接调用分析部 107 分析图 13 的程序 111 的结果的虚函数直接调用信息 115 的图。

接着，说明虚函数调用变换处理(S108)的细节。图 19 是虚函数调用

变换处理(S108)的详细流程图。这里，作为输入到程序变换装置 100 中的程序 111，考虑图 13 所示的程序 111。

虚函数调用变换处理(S108)包含从 d1 到 d8 的步骤。以下按记号的顺序来进行说明。

在步骤 d1 中，虚函数调用变换部 108 对程序 111 中的包含虚函数 F1 的调用的所有语句 S1 重复以下的处理。在图 13 中，“this->g();”、“this->h();”等成为处理对象。

在步骤 d2 中，虚函数调用变换部 108 检查语句 S1 是否被包含在虚函数定义 F2 中。在语句 S1 被包含在虚函数定义 F2 中的情况下，将处理转移到步骤 d3；而在不包含在虚函数定义 F2 中的情况下，将处理转移到步骤 d8。图 13 的语句 “this->g();” 被包含在虚函数 “A::f” 中，所以步骤 d2 的判定为“是”，将处理转移到步骤 d3。同样，语句 “this->h();” 被包含在虚函数 “B::g” 中，所以步骤 d2 的判定为“是”，将处理转移到步骤 d3。此外，语句 “obj_c.g();” 被包含在函数 t 中，但是函数 t 不是虚函数，所以步骤 d2 的判定为“否”，将处理转移到步骤 d8。

在步骤 d3 中，虚函数调用变换部 108 检查语句 S1 是否是 this 指针的调用。在语句 S1 是 this 指针的调用的情况下，将处理转移到步骤 d4；而在不是 this 指针的调用的情况下，将处理转移到步骤 d8。在图 13 的语句 “this->g();” 及语句 “this->h();” 中，虚函数 g 及 h 的调用是 this 指针进行的，所以步骤 d3 的判定为“是”，将处理转移到步骤 d4。这里，说明 this 指针。如上所述，所谓 this 指针，表示在成员函数的内部指向启动了该成员函数的对象的特别的指针。例如，通过图 13 的函数 t 内的成员函数调用 “obj_c.g();”，来执行类 B 的成员函数 g。此时，类 B 的成员函数 g 的 this 指针指向启动了成员函数 g 的对象 obj_c。

在步骤 d4 中，虚函数调用变换部 108 参照类的继承关系信息 113 来检查虚函数 F2 所属的类 C1 是否具有派生类。在类 C1 具有派生类的情况下，将处理转移到步骤 d5；而在没有派生类的情况下，将处理转移到步骤 d7。在图 13 的虚函数 “A::f” 中，参照类的继承关系信息 113 可知

f 所属的类 A 具有派生类，所以步骤 d4 的判定为“是”，将处理转移到步骤 d5。同样，在虚函数“B::g”中，可知 g 所属的类 B 具有派生类，所以步骤 d4 的判定为“是”。

在步骤 d5 中，虚函数调用变换部 108 检查是否在类 C1 的所有直接派生类 C2 中重定义了虚函数 F2。在所有直接派生类 C2 中重定义了虚函数 F2 的情况下，将处理转移到步骤 d6；而在未重定义的情况下，将处理转移到步骤 d8。在图 13 中，参照虚函数重定义信息 114 可知，在类 A 的所有直接派生类 B 中重定义了虚函数 f，所以步骤 d5 的判定为“是”，将处理转移到步骤 d6。而在类 B 的直接派生类 D 中重定义了虚函数 g，但是在直接派生类 C 中未重定义虚函数 g，所以步骤 d5 的判定为“否”，将处理转移到步骤 d8。

在步骤 d6 中，虚函数调用变换部 108 检查类 C1 的所有派生类 C3 的对象是否不直接调用虚函数 F2。在不直接调用虚函数 F2 的情况下，将处理转移到步骤 d7；而在直接调用了的情况下，将处理转移到步骤 d8。考虑图 13 的程序 111。参照虚函数直接调用信息 115 可知，类 A 的所有派生类 B、C 及 D 的对象不直接调用虚函数 f，所以步骤 d6 的判定为“是”，将处理转移到步骤 d7。

在步骤 d7 中，虚函数调用变换部 108 将语句 S1 变换为函数 F1 的直接调用。对图 13 的语句“this->g();”，将虚函数 g 变换为直接调用，变换为与描述成“this->A::g();”的情况同等的代码。

在步骤 d8 中，虚函数调用变换部 108 将处理转移到步骤 d1，重复循环的处理。

图 20 是虚函数调用变换部 108 变换图 13 的程序 111 的结果所得到的代码的图。在该代码中，直接调用了虚函数 A::g。

如上所述，向本实施方式的程序变换装置 100 输入了图 13 所示的程序 111 的结果，能够将包含虚函数调用的语句“this->g();”变换为函数的直接调用。因此，能够提高执行目标程序 112 时的执行速度。

其中，在上述实施方式中，类继承关系分析处理(S104)、虚函数重定

义分析处理(S105)、虚函数直接调用分析处理(S107)、虚函数调用变换处理(S108)分别只执行了一次，但是也可以在必要时重复多次。

此外，在虚函数及虚函数所属的类满足特定条件的情况下，能够在该虚函数定义中确定 this 指针的类型，但是在进行类对象的生成处理的构造函数、或进行删除处理的析构函数中，this 指针的类型能够确定为该构造函数或析构函数所属的类。因此，也可以将构造函数或析构函数中描述的 this 指针的虚函数调用变换为函数的直接调用。

此外，在能够根据程序的性质或执行结果来清楚地确定 this 指针的类型的情况下，也可以根据 pragma 指令、选项指定及程序的执行历史信息等，将虚函数调用变换为函数的直接调用。

[实施方式 2]

在本实施方式中，虚函数调用变换部 108 执行的处理与实施方式 1 不同。

在本实施方式中，在满足以下的条件(1)至条件(4)的情况下，能够确定 this 指针的类型，所以将虚函数 F1 的调用变换为直接函数调用。

即，

- (1) 在虚函数 F2 的定义中描述了虚函数 F1。
- (2) this 指针调用了虚函数 F1。
- (3) 虚函数 F2 所属的类 C1 没有派生类，或者在重定义了虚函数 F1 的派生类 C2 中也重定义了虚函数 F2。
- (4) 类 C2 以外的派生类的对象不直接调用虚函数 F2。

图 21 是说明实施方式 2 的程序变换装置根据上述 4 个条件执行的处理的概略的图。即，考虑给出了图 21(a)所示的类的定义、而且给出了图 21(b)所示的程序的情况。

在设虚函数 g 为虚函数 F1、设虚函数 f 为虚函数 F2 的情况下，图 21(b)的语句 “this->g()” 满足条件(1)及条件(2)。即，在虚函数 F2(虚函数 f)的定义中描述了虚函数 F1(虚函数 g)，所以满足条件(1)。此外，this 指针调用了虚函数 F1(虚函数 g)，所以满足条件(2)。

此外，考虑条件(3)。如图 21(a)所示，在重定义了虚函数 F1(虚函数 g)的派生类 C2(类 C)中也重定义了虚函数 F2(虚函数 f)。因此，满足条件(3)。

再者，考虑条件(4)。在程序中没有类 B 以外的 A::f(void)的直接调用。因此，类 C2(类 C)以外的派生类(类 B、D)的对象不直接调用虚函数 F2(虚函数 f)。因此，满足条件(4)。

因此，在这种情况下，能够将图 21(b)所示的虚函数 g 的调用，如图 21(c)所示变换为类 A 的虚函数 g 的直接调用。

以下，参照程序例来说明实施方式 2 的程序变换装置 100 执行的处理。

如图 11 所示，首先，输入部 102 从程序存储部 101 输入所有程序 111(S102)，语法分析部 103 进行程序的语法树或符号表的生成等(S103)。

接着，类继承关系分析部 104 分析程序 111 中包含的类及类间的继承关系，将分析结果记录到类的继承关系信息 113 中(S104)。类继承关系分析处理(S104)与图 12 所示的相同。类继承关系分析处理(S104)分析图 13 的程序 111 的结果——类的继承关系信息 113 示于图 22。

接着，虚函数重定义分析部 105 分析基类的虚函数的重定义状态，将分析结果记录到虚函数重定义信息 114 中(S105)。虚函数重定义分析处理(S105)与图 15 所示的相同。虚函数重定义分析处理(S105)分析图 13 的程序 111 的结果——虚函数重定义信息 114 示于图 23。

接着，虚函数直接调用分析部 107 分析要直接调用的虚函数，并将分析结果记录到虚函数直接调用信息 115 中(S107)。虚函数直接调用分析处理(S107)与图 17 所示的相同。虚函数直接调用分析处理(S107)分析图 13 的程序 111 的结果——虚函数直接调用信息 115 示于图 24。

接着，参照图 25 来说明虚函数调用变换部 108 执行的处理，以将该处理适用于图 13 所示的程序 111 的情况为例。

虚函数调用变换部 108 执行的虚函数调用变换处理(S108)包含从 e1 到 e8 的步骤。以下按记号的顺序来进行说明。

在步骤 e1 中，虚函数调用变换部 108 对程序中的包含虚函数调用的所有语句 S1 重复以下的处理。在图 13 中处理 “this->g();”、“this->h();” 等。

在步骤 e2 中，虚函数调用变换部 108 检查语句 S1 是否被包含在虚函数定义 F2 中。在语句 S1 被包含在虚函数定义 F2 中的情况下，将处理转移到步骤 e3；而在不包含在虚函数定义 F2 中的情况下，将处理转移到步骤 e8。图 13 的语句 “this->g();” 被包含在虚函数 “A::f” 中，所以步骤 e2 的判定为 “是”，将处理转移到步骤 e3。同样，语句 “this->h();” 被包含在虚函数 “B::g” 中，所以步骤 e2 的判定为 “是”，将处理转移到步骤 e3。而语句 “obj_c.g();” 被包含在函数 t 中，函数 t 不是虚函数，所以步骤 e2 的判定为 “否”，将处理转移到步骤 e8。

在步骤 e3 中，虚函数调用变换部 108 检查语句 S1 是否是 this 指针的调用。在语句 S1 是 this 指针的调用的情况下，将处理转移到步骤 e4；而在不是对 this 指针的调用的情况下，将处理转移到步骤 e8。在图 13 的语句 “this->g();” 及语句 “this->h();” 中，虚函数 g 及 h 的调用是 this 指针进行的，所以步骤 e3 的判定为 “是”，将处理转移到步骤 e4。

在步骤 e4 中，虚函数调用变换部 108 参照类的继承关系信息 113，来检查虚函数 F2 所属的类 C1 是否具有派生类。在类 C1 具有派生类的情况下，将处理转移到步骤 e5；而在没有派生类的情况下，将处理转移到步骤 e7。在图 13 的虚函数 “A::f” 中，参照类的继承关系信息 113 可知，f 所属的类 A 具有派生类，所以步骤 e4 的判定为 “是”，将处理转移到步骤 e5。同样，在虚函数 “B::g” 中，可知 g 所属的类 B 具有派生类，所以步骤 e4 的判定为 “是”。

在步骤 e5 中，虚函数调用变换部 108 检查是否在类 C1 的派生类中的、重定义了虚函数 F1 的派生类 C2 中重定义了虚函数 F2。在重定义了虚函数 F2 的情况下，将处理转移到步骤 e6；而在未重定义的情况下，将处理转移到步骤 e8。在图 13 中，参照虚函数重定义信息 114 可知，重定义了虚函数 “A::g” 的类 A 的派生类只有类 B，同时可知在类 B 中重定

义了虚函数 f。因此，步骤 e5 的判定为“是”，将处理转移到步骤 e6。而重定义了虚函数“A::h”的类 B 的派生类只有类 D，同时在类 D 中重定义了虚函数 g，所以步骤 e5 的判定为“是”。

其中，一旦在派生类 C2 中重定义了虚函数 F1，则在步骤 e5 的处理中无需将继承了 C2 的派生类作为搜索对象。这是因为，即使未在继承了 C2 的派生类中重定义虚函数 F1，从 C2 的派生类调用的 F1 也是类 C2 的 F1，类 C1 的虚函数 F1 必定处于重定义了的状态。

在步骤 e6 中，虚函数调用变换部 108 检查类 C2 及 C2 的所有派生类的对象是否不直接调用虚函数 F2。在不直接调用虚函数 F2 的情况下，将处理转移到步骤 e7；而在直接调用了的情况下，将处理转移到步骤 e8。在图 13 中，参照虚函数直接调用信息 115 可知，类 B 及类 B 的派生类——类 C、D 的对象不直接调用虚函数 f，所以步骤 e6 的判定为“是”，将处理转移到步骤 e7。同样，只有类 C 的对象直接调用虚函数 g，类 D 的对象不直接调用，所以步骤 e6 的判定为“是”。

在步骤 e7 中，虚函数调用变换部 108 将语句 S1 变换为函数 F1 的直接调用。对图 13 的语句“this->g();”，将虚函数 g 变换为直接调用，变换为与描述成“this->A::g();”的情况同等的代码。同样，将“this->h();”变换为与描述成“this->B::h();”的情况同等的代码。

在步骤 e8 中，虚函数调用变换部 108 将处理转移到步骤 e1，重复循环的处理。

虚函数调用变换部 108 变换图 13 的程序 111 的结果所得到的代码示于图 26。在该代码中，直接调用了虚函数 A::g() 和虚函数 B::h()。

如上所述，向本实施方式的程序变换装置 100 输入了图 13 所示的程序 111 的结果是，能够将包含虚函数调用的语句“this->g();”及语句“this->h();”变换为函数的直接调用。因此，能够提高执行目标程序 112 时的执行速度。

其中，在上述实施方式中，类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)、虚函数调用变换处

理(S108)分别只执行了一次，但是也可以在必要时重复多次。

此外，在虚函数及虚函数所属的类满足特定的条件的情况下，能够在该虚函数定义中确定 this 指针的类型，但是在进行类对象的生成处理的构造函数、或进行删除处理的析构函数中，this 指针的类型能够确定为该构造函数或析构函数所属的类。因此，也可以将构造函数或析构函数中描述的 this 指针的虚函数调用变换为函数的直接调用。

此外，在能够根据程序的性质或执行结果来清楚地确定 this 指针的类型的情况下，也可以根据 pragma 指令、选项指定及程序的执行历史信息等，将虚函数调用变换为函数的直接调用。

[实施方式 3]

在本实施方式中，在满足以下的条件(1)至条件(4)的情况下，能够确定 this 指针的类型，所以将虚函数 F1 的调用变换为直接函数调用。

即，

- (1) 在虚函数 F2 的定义中描述了虚函数 F1。
- (2) this 指针调用了虚函数 F1。
- (3) 虚函数 F2 所属的类 C1 的派生类的对象不直接调用虚函数 F2。
- (4) 复制所有不在派生类中重定义的虚函数。

图 27 是说明实施方式 3 的程序变换装置根据上述 4 个条件执行的处理的概略的图。即，考虑给出了图 27(a)所示的类的定义、和图 27(c)所示的程序的情况。

在设虚函数 g 为虚函数 F1、设虚函数 f 为虚函数 F2 的情况下，图 27(c)的语句 “this->g()” 满足条件(1)及条件(2)。即，在虚函数 F2(虚函数 f)的定义中描述了虚函数 F1(虚函数 g)，所以满足条件(1)。此外，this 指针调用了虚函数 F1(虚函数 g)，所以满足条件(2)。

此外，考虑条件(3)。在程序中不存在 A::f(void)的直接调用。因此，虚函数 F2(虚函数 f)所属的类 C1(类 A)的派生类(类 B、C、D)的对象不直接调用虚函数 F2(虚函数 f)。因此，满足条件(3)。

再者，复制所有未在派生类(类 B、C、D)中重定义的虚函数。即，如图 27(b)所示，在类 B 中复制虚函数 f 及虚函数 g，并在类 D 中复制虚函数 g。由此，满足条件(4)。

因此，在这种情况下，能够将图 27(c)所示的虚函数 g 的调用如图 27(d)所示变换为类 A 的虚函数 g 的直接调用。

[程序变换装置的结构]

图 28 是本发明实施方式 3 的程序变换装置 200 的结构的图。程序变换装置 200 包括：输入部 102、语法分析部 103、类继承关系分析部 104、虚函数重定义分析部 105、分析信息存储部 206、虚函数直接调用分析部 107、虚函数调用变换判定部 201、虚函数定义复制部 202、虚函数调用变换部 203、以及输出部 109 构成。

其中，程序存储部 101、输入部 102、语法分析部 103、类继承关系分析部 104、虚函数重定义分析部 105、虚函数直接调用分析部 107、输出部 109 及生成代码存储部 110 进行与图 10 所示的实施方式 1 的程序变换装置 100 的各部同样的处理。因此，这里不重复其详细说明。

虚函数调用变换判定部 201 参照分析信息存储部 106 中保存的类的继承关系信息 113 及虚函数直接调用信息 115，判定能否将虚函数调用变换为函数的直接调用，将与包含能够变换为直接调用的虚函数调用的语句有关的信息(以下称为“可直接调用语句信息”。)保存到分析信息存储部 106 中。

虚函数定义复制部 202 参照分析信息存储部 106 中保存的类的继承关系信息 113 及虚函数重定义信息 114，复制未在派生类中重定义的虚函数，变换为在派生类中重定义了的状态。

虚函数调用变换部 203 将分析信息存储部 106 中保存的可直接调用语句信息 116 中包含的语句，变换为函数的直接调用。

分析信息存储部 106 是存储各步骤中分析出的信息的存储装置。即，分析信息存储部 106 是存储类的继承关系信息 113、虚函数重定义信息 114、虚函数直接调用信息 115 及可直接调用语句信息 116 的存储装置。

[程序变换装置执行的处理的概要]

图 29 是本实施方式的程序变换装置 200 执行的处理的流程图。

其中，输入处理(S102)、语法分析处理(S103)、类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)及输出处理(S109)，与图 11 所示的程序变换装置 100 执行的对应的处理相同。因此，这里不重复其详细说明。

在虚函数调用变换判定处理(S201)中，虚函数调用变换判定部 201 参照分析信息存储部 106 中保存的类的继承关系信息 113 及虚函数直接调用信息 115，判定能否将虚函数调用变换为函数的直接调用，并将可直接调用语句信息 116 保存到分析信息存储部 206 中。

在虚函数定义复制处理(S202)中，虚函数定义复制部 202 参照分析信息存储部 106 中保存的类的继承关系信息 113 及虚函数重定义信息 114，复制未在派生类中重定义的虚函数，变换为在派生类中重定义了的状态。

在虚函数调用变换处理(S203)中，虚函数调用变换部 203 将分析信息存储部 106 中保存的可直接调用语句信息 116 中包含的语句，变换为函数的直接调用。

[程序变换装置执行的处理的细节]

接着，参照图 30 所示的程序例 111 来说明程序变换装置 200 执行的处理的细节。

首先，输入部 102 从程序存储部 101 输入所有程序 111(S102)，语法分析部 103 进行程序 111 的语法树和符号表的生成等(S103)。

接着，类继承关系分析部 104 分析程序 111 中包含的类及类间的继承关系，将分析结果记录到类的继承关系信息 113 中(S104)。类继承关系分析处理(S104)与图 12 所示的相同。类继承关系分析处理(S104)分析图 30 的程序 111 的结果——类的继承关系信息 113 示于图 31。

接着，虚函数重定义分析部 105 分析基类的虚函数的重定义状态，将分析结果记录到虚函数重定义信息 114 中(S105)。虚函数重定义分析处理(S105)与图 15 所示的相同。虚函数重定义分析处理(S105)分析图 30 的

程序 111 的结果——虚函数重定义信息 114 示于图 32。

接着，虚函数直接调用分析部 107 分析直接调用的虚函数，将分析结果记录到虚函数直接调用信息 115 中(S107)。虚函数直接调用分析处理(S107)与图 17 所示的相同。虚函数直接调用分析处理(S107)分析图 30 的程序 111 的结果——虚函数直接调用信息 115 示于图 33。

接着，参照图 34 来说明虚函数调用变换判定部 201 执行的处理，以将该处理适用于图 30 所示的程序 111 的情况为例。

虚函数调用变换判定部 201 执行的虚函数调用变换判定处理(S201)包含从 f1 到 f6 的步骤。以下按记号的顺序来进行说明。

在步骤 f1 中，虚函数调用变换判定部 201 对程序 111 中的包含虚函数调用的所有语句 S1 重复以下的处理。在图 30 中处理 “this->g();”、“this->h();” 等。

在步骤 f2 中，虚函数调用变换判定部 201 检查语句 S1 是否被包含在虚函数定义 F2 中。在语句 S1 被包含在虚函数定义 F2 中的情况下，将处理转移到步骤 f3，而在不包含在虚函数定义 F2 中的情况下，将处理转移到步骤 f6。图 30 的语句 “this->g();” 被包含在虚函数 “A::f” 中，所以步骤 f2 的判定为“是”，将处理转移到步骤 f3。同样，语句 “this->h();” 被包含在虚函数 “B::g” 中，所以步骤 f2 的判定为“是”，将处理转移到步骤 f3。而语句 “obj_c.g();” 被包含在函数 t 中，函数 t 不是虚函数，所以步骤 f2 的判定为“否”，将处理转移到步骤 f6。

在步骤 f3 中，虚函数调用变换判定部 201 检查语句 S1 是否是 this 指针的调用。在语句 S1 是 this 指针的调用的情况下，将处理转移到步骤 f4，而在不是对 this 指针的调用的情况下，将处理转移到步骤 f6。在图 30 的语句 “this->g();” 及语句 “this->h();” 中，this 指针进行虚函数 g 及 h 的调用，所以步骤 f3 的判定为“是”，将处理转移到步骤 f4。

在步骤 f4 中，虚函数调用变换判定部 201 检查虚函数 F2 所属的类 C1 的所有派生类 C3 的对象是否不直接调用虚函数 F2。在未直接调用虚函数 F2 的情况下，将处理转移到步骤 f5，而在直接调用了的情况下，将

处理转移到步骤 f6。在图 30 中，参照虚函数直接调用信息 115 可知，类 A 的所有派生类 B、C 及 D 的对象不直接调用虚函数“A::f”，所以步骤 f4 的判定为“是”，将处理转移到步骤 f5。而检查虚函数“B::g”的直接调用可知，类 C 的对象直接调用了虚函数“B::g”，所以步骤 f4 的判定为“否”，将处理转移到步骤 f6。

在步骤 f5 中，虚函数调用变换判定部 201 将语句 S1 添加到可直接调用语句信息 116 中。在图 30 中，将语句“this->g();”添加到可直接调用语句信息 116 中。

在步骤 f6 中，虚函数调用变换判定部 201 将处理转移到步骤 f1，重复循环的处理。

虚函数调用变换判定部 201 通过虚函数调用变换判定处理(S201)分析图 30 的程序 111 的结果——可直接调用语句信息 116 示于图 35。

接着，参照图 36 来说明虚函数定义复制部 202 执行的虚函数定义复制处理(S202)，以将该处理适用于图 30 所示的程序 111 的情况为例。

虚函数定义复制部 202 执行的虚函数定义复制处理(S202)包含从 g1 到 g7 的步骤。以下按记号的顺序来进行说明。

在步骤 g1 中，虚函数定义复制部 202 检查可直接调用语句信息 116 是否是空集。在是空集的情况下，结束虚函数定义复制处理(S202)，而在不是空集的情况下，将处理转移到步骤 g2。在图 30 的程序 111 中，可直接调用语句信息 116 不是空集，所以步骤 g1 的判定为“否”，将处理转移到步骤 g2。

在步骤 g2 中，虚函数定义复制部 202 对所有程序 111 中包含的所有类 C1 重复以下的处理。即，对图 30 的所有程序 111 中包含的类 A、B、C 及 D 重复。

在步骤 g3 中，虚函数定义复制部 202 检查类 C1 是否具有基类 C2。在类 C1 具有基类 C2 的情况下，将处理转移到步骤 g4；而在没有基类 C2 的情况下将处理转移到步骤 g7。在图 30 的类 A 的情况下，类 A 没有基类，所以步骤 g3 的判定为“否”，将处理转移到步骤 g7。而在类 B 的

情况下，作为基类而具有类 A，所以步骤 g3 的判定为“是”，将处理转移到步骤 g4。类 C 及 D 也分别具有基类 B，所以步骤 g3 的判定为“是”，将处理转移到步骤 g4。

在步骤 g4 中，虚函数定义复制部 202 检查在类 C2 的虚函数中是否存在未在类 C1 中重定义的函数 F1。在存在未在类 C1 中重定义的函数 F1 的情况下，将处理转移到步骤 g5；而在不存在的情况下，将处理转移到步骤 g7。在图 30 的类 B 的情况下，参照虚函数重定义信息 114 可知，未重定义基类 A 的虚函数 f 及 h，所以步骤 g4 的判定为“是”，将处理转移到步骤 g5。同样，在类 B、C 及 D 的情况下也分别有未重定义基类的虚函数的情况，所以步骤 g4 的判定为“是”。

在步骤 g5 中，虚函数定义复制部 202 复制类 C2 的虚函数 F1 来生成虚函数 F2，并以在类 C1 中重定义了的形式变换类 C1 的定义。在图 30 的类 B 的情况下，复制基类 A 的虚函数 f 及 h 来生成 f' 及 h'，并以在类 B 中重定义了的形式变换类 B 的定义。在类 C 及 D 的情况下，也同样复制基类 B 的虚函数，并变换类的定义。

在步骤 g6 中，虚函数定义复制部 202 将类 C1 的虚函数表的函数 F1 一栏置换为函数 F2。在图 30 的类 B 的情况下，类 B 的虚函数表的函数 f 及 h 一栏分别是“A::f(void)”、“A::h(void)”，改写为复制的函数“B::f'(void)”“B::h'(void)”。同样，将类 C、类 D 的虚函数表一栏改写为复制的函数。

在步骤 g7 中，虚函数定义复制部 202 将处理转移到步骤 g2，重复循环的处理。

虚函数定义复制处理(S202)变换图 30 的程序 111 的结果的程序及各类的虚函数表分别示于图 37 及图 38。

接着，参照图 39 来说明虚函数调用变换部 203 执行的虚函数调用变换处理(S203)，以将该处理适用于图 30 所示的程序 111 的情况为例。

虚函数调用变换部 203 执行的虚函数调用变换处理(S203)包含从 h1 到 h3 的步骤。以下按记号的顺序来进行说明。

在步骤 h1 中，虚函数调用变换部 203 对可直接调用语句信息 116 中包含的所有语句 S1 重复以下的处理。在图 30 的程序 111 中，对可直接调用语句信息 116 中包含的语句 “this->g()” 执行以下的处理。

在步骤 h2 中，虚函数调用变换部 203 将语句 S1 中包含的虚函数调用变换为函数的直接调用。在图 30 中，将语句 “this->g()” 的虚函数调用变换为函数的直接调用，变换为与描述成 “this->A::g();” 的情况同等的代码。

在步骤 h3 中，虚函数调用变换部 203 将处理转移到步骤 h1，重复循环的处理。

虚函数调用变换处理(S203)变换图 30 的程序 111 的结果所得到的代码示于图 40。

如上所述，向本实施方式的程序变换装置 200 适用了图 30 所示的程序 111 的结果是，能够将包含虚函数调用的语句 “this->g();” 变换为函数的直接调用。因此，能够提高执行目标程序 112 时的执行速度。

其中，在上述实施方式中，类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)、虚函数调用变换判定处理(S201)、虚函数定义复制处理(S202)、虚函数调用变换处理(S203) 分别只执行了一次，但是也可以在必要时重复多次。

此外，在虚函数及虚函数所属的类满足特定的条件的情况下，能够在该虚函数定义中确定 this 指针的类型，但是在进行类对象的生成处理的构造函数、或进行删除处理的析构函数中，this 指针的类型能够确定为该构造函数或析构函数所属的类。因此，也可以将构造函数或析构函数中描述的 this 指针的虚函数调用变换为函数的直接调用。

此外，在能够根据程序的性质或执行结果来清楚地确定 this 指针的类型的情况下，也可以根据 pragma 指令、选项指定及程序的执行历史信息等，将虚函数调用变换为函数的直接调用。

[实施方式 4]

在本实施方式中，虚函数调用变换判定部 201 及虚函数定义复制部

202 执行的处理与实施方式 3 不同。后面将描述虚函数调用变换判定部 201 及虚函数定义复制部 202 分别执行的虚函数调用变换判定处理(S201)及虚函数定义复制处理(S202)。

此外，分析信息存储部 206 还存储复制虚函数信息(未图示)。后面将描述复制虚函数信息。

在本实施方式中，在满足以下的条件(1)至条件(4)的情况下，能够确定 this 指针的类型，所以将虚函数 F1 的调用变换为直接函数调用。

即，

- (1) 在虚函数 F2 的定义中描述了虚函数 F1。
- (2) this 指针调用了虚函数 F1。
- (3) 虚函数 F2 所属的类 C1 以外的派生类的对象不直接调用虚函数 F2。

(4) 复制所有未在类 C1 的直接派生类中重定义的虚函数 F2。

图 41 是说明实施方式 4 的程序变换装置根据上述 4 个条件执行的处理的概略的图。即，考虑给出了图 41(a)所示的类的定义、和图 41(c)所示的程序的情况。

在设虚函数 g 为虚函数 F1、设虚函数 f 为虚函数 F2 的情况下，图 41(c)的语句 “this->g()” 满足条件(1)及条件(2)。即，在虚函数 F2(虚函数 f)的定义中描述了虚函数 F1(虚函数 g)，所以满足条件(1)。此外，this 指针调用了虚函数 F1(虚函数 g)，所以满足条件(2)。

此外，考虑条件(3)。在程序中不存在 A::f(void)的直接调用。因此，虚函数 F2(虚函数 g)所属的类 C1(类 A)以外的派生类(类 B、C、D)的对象不直接调用虚函数 F2(虚函数 g)。因此，满足条件(3)。

再者，复制所有未在类 C1(类 A)的直接派生类(类 B)中重定义的虚函数 F2(虚函数 f)。即，如图 41(b)所示，在类 B 中复制虚函数 f。由此，满足条件(4)。

因此，在这种情况下，如图 41(d)所示，能够将图 41(c)所示的虚函数 g 的调用变换为类 A 的虚函数 g 的直接调用。

以下，参照程序例来说明实施方式 4 的程序变换装置 200 执行的处理。

首先，输入部 102 从程序存储部 101 输入所有程序 111(S102)，语法分析部 103 进行程序的语法树和符号表的生成等(S103)。

接着，类继承关系分析部 104 分析程序中包含的类及类间的继承关系，并将分析结果记录到类的继承关系信息 113 中(S104)。类继承关系分析处理(S104)与图 12 所示的相同。类继承关系分析处理(S104)分析图 30 的程序 111 的结果——类的继承关系信息 113 示于图 42。

接着，虚函数重定义分析部 105 分析基类的虚函数的重定义状态，将分析结果记录到虚函数重定义信息 114 中(S105)。虚函数重定义分析处理(S105)与图 15 所示的相同。虚函数重定义分析处理(S105)分析图 30 的程序 111 的结果——虚函数重定义信息 114 示于图 43。

接着，虚函数直接调用分析部 107 分析直接调用的虚函数，并将分析结果记录到虚函数直接调用信息 115 中(S107)。虚函数直接调用分析处理(S107)与图 17 所示的相同。虚函数直接调用分析处理(S107)分析图 30 的程序 111 的结果——虚函数直接调用信息 115 示于图 44。

接着，参照图 45 来说明虚函数调用变换判定部 201 执行的虚函数调用变换处理(S201)的细节，以将该处理适用于图 30 所示的程序 111 的情况为例。

虚函数调用变换判定处理(S201)包含从 j1 到 j7 的步骤。以下按记号的顺序来进行说明。

在步骤 j1 中，虚函数调用变换判定部 201 对程序 111 中的包含虚函数调用的所有语句 S1 重复以下的处理。在图 30 中，“this->g();”、“this->h();”等成为处理对象。

在步骤 j2 中，虚函数调用变换判定部 201 检查语句 S1 是否被包含在虚函数定义 F2 中。在语句 S1 被包含在虚函数定义 F2 中的情况下将处理转移到步骤 j3，而在不包含在虚函数定义 F2 中的情况下将处理转移到步骤 j7。图 30 的语句 “this->g();” 被包含在虚函数 A::f 中，所以步骤 j2 的

判定为“是”，将处理转移到步骤 j3。同样，语句“this->h();”被包含在虚函数“B::g”中，所以步骤 j2 的判定为“是”，将处理转移到步骤 j3。而语句“obj_c.g();”被包含在函数 t 中，函数 t 不是虚函数，所以步骤 j2 的判定为“否”，将处理转移到步骤 j7。

在步骤 j3 中，虚函数调用变换判定部 201 检查语句 S1 是否是 this 指针的调用。在语句 S1 是 this 指针的调用的情况下，将处理转移到步骤 j4；而在不是 this 指针的调用的情况下，将处理转移到步骤 j7。在图 30 的语句“this->g();”及语句“this->h();”中，虚函数 g 及 h 的调用是 this 指针进行的，所以步骤 j3 的判定为“是”，将处理转移到步骤 j4。

在步骤 j4 中，虚函数调用变换判定部 201 检查虚函数 F2 所属的类 C1 的所有派生类 C3 的对象是否不直接调用虚函数 F2。在不直接调用虚函数 F2 的情况下将处理转移到步骤 j5，而在直接调用了的情况下，将处理转移到步骤 j7。在图 30 中，参照虚函数直接调用信息 115 可知，类 A 的所有派生类 B、C 及 D 的对象不直接调用虚函数“A::f”，所以步骤 j4 的判定为“是”，将处理转移到步骤 j5。而检查虚函数“B::g”的直接调用可知，类 C 的对象直接调用了虚函数“B::g”，所以步骤 j4 的判定为“否”，将处理转移到步骤 j7。

在步骤 j5 中，虚函数调用变换判定部 201 将语句 S1 添加到可直接调用语句信息 116 中。在图 30 中，将语句“this->g();”添加到可直接调用语句信息 116 中。

在步骤 j6 中，虚函数调用变换判定部 201 将类 C1 的所有直接基类和虚函数 F2 的对子 P1 作为复制虚函数信息而添加到分析信息存储部 206 中。在图 30 中，类 A 的直接基类只有类 B，所以将类 B 和 A::f 的对子添加到复制虚函数信息中。在直接基类有多个的情况下，将多个对子添加到复制虚函数信息中。

在步骤 j7 中，虚函数调用变换判定部 201 将处理转移到步骤 j1，重复循环的处理。

虚函数调用变换判定处理(S201)分析图 30 的程序 111 的结果——可

直接调用语句信息 116 及复制虚函数信息分别示于图 46 及图 47。

接着，参照图 48 来说明虚函数定义复制部 202 执行的虚函数定义复制处理(S202)，以将该处理适用于图 30 所示的程序 111 的情况为例。

虚函数定义复制处理(S202)包含从 k1 到 k6 的步骤。以下按记号的顺序来进行说明。

在步骤 k1 中，虚函数定义复制部 202 检查可直接调用语句信息 116 是否是空集。在是空集的情况下，结束虚函数定义复制处理(S202)，而在不是空集的情况下，将处理转移到步骤 k2。在图 30 的程序 111 中，可直接调用语句信息 116 不是空集，所以步骤 k1 的判定为“否”，将处理转移到步骤 k2。

在步骤 k2 中，虚函数定义复制部 202 对复制虚函数信息的所有对子 P1 重复以下的处理。在图 30 中，对类 B 和虚函数“A::f”的对子重复以下的处理。

在步骤 k3 中，虚函数定义复制部 202 检查是否在对子 P1 的类 C1 中重定义了对子 P1 的虚函数 F1。在未重定义的情况下将处理转移到步骤 k4，而在重定义了的情况下将处理转移到步骤 k6。在图 30 的对子 P1 的类 B 的情况下，参照虚函数重定义信息 114 可知，未在类 B 中重定义对子 P1 的虚函数“A::f”，所以步骤 k3 的判定为“是”，将处理转移到步骤 k4。

在步骤 k4 中，虚函数定义复制部 202 复制虚函数 F1 来生成虚函数 F2，并以在类 C1 中重定义了的形式变换类 C1 的定义。在图 30 的类 B 的情况下，复制虚函数“A::f”来生成 f，并以在类 B 中重定义了的形式变换类 B 的定义。

在步骤 k5 中，虚函数定义复制部 202 将登录有所有类 C2 的虚函数表的函数 F1 的一栏置换为函数 F2。在图 30 中，将类 B、C 及 D 的虚函数表的函数 f “A::f(void)”一栏改写为复制的函数 “B::f(void)”。

在步骤 k6 中，虚函数定义复制部 202 将处理转移到步骤 k2，重复循环的处理。

通过虚函数定义复制部 202 执行的虚函数定义复制处理(S202)而变换图 30 的程序 111 的结果——程序及各类的虚函数表分别示于图 49 及图 50。

接着，虚函数调用变换部 203 将虚函数调用变换为函数的直接调用(S203)。虚函数调用变换处理(S203)与图 39 所示的相同。虚函数调用变换处理(S203)变换图 30 的程序 111 的结果所得到的代码示于图 51。

如上所述，向本实施方式的程序变换装置 200 输入了图 30 所示的程序的结果是，能够将包含虚函数调用的语句 “this->g();” 变换为函数的直接调用。因此，能够提高执行目标程序 112 时的执行速度。此外，程序变换装置 200 复制所需最低限度的虚函数。因此，也能够减少目标程序 112 的代码长度的增加。

其中，在上述实施方式中，类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)、虚函数调用变换判定处理(S201)、虚函数定义复制处理(S202)、虚函数调用变换处理(S203)分别只执行一次，但是也可以在必要时重复多次。

此外，在虚函数及虚函数所属的类满足特定的条件的情况下，能够在该虚函数定义中确定 this 指针的类型，但是在进行类对象的生成处理的构造函数、或进行删除处理的析构函数中，this 指针的类型能够确定为该构造函数或析构函数所属的类。因此，也可以将构造函数或析构函数中描述的 this 指针的虚函数调用变换为函数的直接调用。

此外，在能够根据程序的性质或执行结果来清楚地确定 this 指针的类型的情况下，也可以根据 pragma 指令、选项指定及程序的执行历史信息等，将虚函数调用变换为函数的直接调用。

[实施方式 5]

本实施方式 5 的程序变换装置 200 的虚函数调用变换判定部 201 进行的处理与实施方式 4 不同。后面将描述虚函数调用变换判定部 201 执行的虚函数调用判定处理(S201)。

在本实施方式中，在满足以下的条件(1)至条件(4)的情况下，能够确

定 this 指针的类型，所以将虚函数 F1 的调用变换为直接函数调用。

即，

- (1) 在虚函数 F2 的定义中描述了虚函数 F1。
- (2) this 指针调用了虚函数 F1。
- (3) 在虚函数 F2 所属的类 C1 的派生类中，未重定义虚函数 F1 的类 C2 以外的类的对象不直接调用虚函数 F2。
- (4) 复制所有未在类 C2 以外的派生类中重定义的虚函数 F2。

图 52 是说明实施方式 5 的程序变换装置根据上述 4 个条件执行的处理的概略的图。即，考虑给出了图 52(a)所示的类的定义、和图 52(c)所示的程序的情况。

在设虚函数 g 为虚函数 F1、设虚函数 f 为虚函数 F2 的情况下，图 52(c)的语句 “this->g()” 满足条件(1)及条件(2)。即，在虚函数 F2(虚函数 f)的定义中描述了虚函数 F1(虚函数 g)，所以满足条件(1)。此外，this 指针调用了虚函数 F1(虚函数 g)，所以满足条件(2)。

此外，考虑条件(3)。在程序中不存在类 B 以外的 A::f(void)的直接调用。因此，在虚函数 F2(虚函数 f)所属的类 C1(类 A)的派生类(类 B、C、D)中、未重定义虚函数 F1(虚函数 g)的类 C2(类 B、D)以外的类(类 C)的对象不直接调用虚函数 F2(虚函数 f)。因此，满足条件(3)。

再者，复制所有未在类 C2(类 B、D)以外的派生类(类 C)中重定义的虚函数 F2(虚函数 f)。即，如图 52(b)所示，在类 C 中复制虚函数 f。由此，满足条件(4)。

因此，在这种情况下，如图 52(d)所示，能够将图 52(c)所示的虚函数 g 的调用变换为类 A 的虚函数 g 的直接调用。

以下，参照程序例来说明实施方式 5 的程序变换装置 200 执行的处理。

首先，输入部 102 从程序存储部 101 输入所有程序 111(S102)，语法分析部 103 进行程序的语法树和符号表的生成等(S103)。

接着，类继承关系分析部 104 分析程序中包含的类及类间的继承关

系，将分析结果记录到类的继承关系信息 113 中(S104)。类继承关系分析处理(S104)的处理与图 12 所示的相同。类继承关系分析处理(S104)分析图 30 的程序 111 的结果——类的继承关系信息 113 示于图 53。

接着，虚函数重定义分析部 105 分析基类的虚函数的重定义状态，将分析结果记录到虚函数重定义信息 114 中(S105)。虚函数重定义分析处理(S105)的处理与图 15 所示的相同。虚函数重定义分析处理(S105)分析图 30 的程序 111 的结果——虚函数重定义信息 114 示于图 54。

接着，虚函数直接调用分析部 107 分析直接调用的虚函数，并将分析结果记录到虚函数直接调用信息 115 中(S107)。虚函数直接调用分析处理(S107)与图 17 所示的相同。虚函数直接调用分析处理(S107)分析图 30 的程序 111 的结果——虚函数直接调用信息 115 示于图 55。

接着，参照图 56 来说明虚函数调用变换判定部 201 执行的虚函数调用变换处理(S201)的细节，以将该处理适用于图 30 所示的程序 111 的情况为例。

虚函数调用变换判定部 201 执行的虚函数调用变换判定处理(S201)包含从 m1 到 m7 的步骤。以下按记号的顺序来进行说明。

在步骤 m1 中，虚函数调用变换判定部 201 对程序 111 中的包含虚函数调用的所有语句 S1 重复以下的处理。在图 30 中处理 “this->g();”、“this->h();” 等。

在步骤 m2 中，虚函数调用变换判定部 201 检查语句 S1 是否被包含在虚函数定义 F2 中。在语句 S1 被包含在虚函数定义 F2 中的情况下，将处理转移到步骤 m3；而在不包含在虚函数定义 F2 中的情况下，将处理转移到步骤 m8。图 30 的语句 “this->g();” 被包含在虚函数 A::f 中，所以步骤 m2 的判定为“是”，将处理转移到步骤 m3。同样，语句 “this->h();” 被包含在虚函数 “B::g” 中，所以步骤 m2 的判定为“是”，将处理转移到步骤 m3。而语句 “obj_c.g();” 被包含在函数 t 中，函数 t 不是虚函数，所以步骤 m2 的判定为“否”，将处理转移到步骤 m8。

在步骤 m3 中，虚函数调用变换判定部 201 检查语句 S1 是否是 this

指针的调用。在语句 S1 是 this 指针的调用的情况下，将处理转移到步骤 m4，而在不是 this 指针的调用的情况下，将处理转移到步骤 m8。在图 30 的语句 “this->g();” 及语句 “this->h();” 中，this 指针进行虚函数 g 及 h 的调用。因此，步骤 m3 的判定为 “是”，将处理转移到步骤 m4。

在步骤 m4 中，虚函数调用变换判定部 201 检查在虚函数 F2 所属的类 C1 的所有派生类中是否有重定义 F1 的类 C2。在有重定义 F1 的类 C2 的情况下，将处理转移到步骤 m5；而在没有的情况下，将处理转移到步骤 m6。在图 30 中，参照虚函数重定义信息 114 可知，在虚函数 “A::f” 所属的类 A 的派生类 B、C 及 D 中重定义虚函数 g 的类是类 B。因此，步骤 m4 的判定为 “是”，将处理转移到步骤 m5。此外，在虚函数 “B::g” 所属的类 B 的派生类 C 及 D 中重定义虚函数 h 的类是类 D，所以步骤 m4 的判定为 “是”，将处理转移到步骤 m5。

在步骤 m5 中，虚函数调用变换判定部 201 检查类 C2 的所有派生类 C3 的对象是否直接调用了虚函数 F2。在直接调用了虚函数 F2 的情况下，将处理转移到步骤 m8，而在未直接调用的情况下，将处理转移到步骤 m6。在图 30 的程序 111 中，参照虚函数直接调用信息 115 可知，未用类 B 的所有派生类 C 及 D 直接调用虚函数 “A::f”，所以步骤 m5 的判定为 “是”，将处理转移到步骤 m6。而类 D 没有派生类，所以派生类不直接调用虚函数 “B::g”，所以步骤 m5 的判定为 “是”。

在步骤 m6 中，虚函数调用变换判定部 201 将语句 S1 添加到可直接调用语句信息 116 中。在图 30 中，将语句 “this->g();” 及语句 “this->h();” 添加到可直接调用语句信息 116 中。

在步骤 m7 中，虚函数调用变换判定部 201 将类 C2 及虚函数 F2 的对子 P1 添加到复制虚函数信息中。在图 30 中，将类 B 和 A::f 的对子、及类 D 和 B::g 的对子添加到复制虚函数信息中。

在步骤 m8 中，虚函数调用变换判定部 201 将处理转移到步骤 m1，重复循环的处理。

虚函数调用变换判定处理(S201)分析图 30 的程序的结果——可直接

调用语句信息 116 及复制虚函数信息分别示于图 57 及图 58。

接着，虚函数定义复制部 202 在必要时复制虚函数的定义，变换类定义及虚函数表(S202)。虚函数定义复制处理(S202)与图 48 所示的相同。虚函数定义复制处理(S202)变换图 30 的程序 111 的结果——程序及各类的虚函数表分别示于图 59 及图 60。

接着，虚函数调用变换部 203 将虚函数调用变换为函数的直接调用(S203)。虚函数调用变换处理(S203)与图 39 所示的相同。虚函数调用变换处理(S203)变换图 30 的程序 111 的结果所得到的代码示于图 61。

如上所述，向本实施方式的程序变换装置 200 输入了图 30 所示的程序 111 的结果是，能够将包含虚函数调用的语句 “this->g();” 及语句 “this->h();” 变换为函数的直接调用。因此，能够提高执行目标程序 112 时的执行速度。此外，程序变换装置 200 复制所需最低限度的虚函数，所以也能够减少目标程序 112 的代码长度的增加。

其中，在上述实施方式中，类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)、虚函数调用变换判定处理(S201)、虚函数定义复制处理(S202)、虚函数调用变换处理(S203)分别只执行了一次，但是也可以在必要时重复多次。

此外，在虚函数及虚函数所属的类满足特定的条件的情况下，能够在该虚函数定义中确定 this 指针的类型，但是在进行类对象的生成处理的构造函数、或进行删除处理的析构函数中，this 指针的类型能够确定为该构造函数或析构函数所属的类。因此，也可以将构造函数或析构函数中描述的 this 指针的虚函数调用变换为函数的直接调用。

此外，在能够根据程序的性质或执行结果来清楚地确定 this 指针的类型的情况下，也可以根据 pragma 指令、选项指定及程序的执行历史信息等，将虚函数调用变换为函数的直接调用。

[实施方式 6]

在实施方式 1 的程序变换装置 100 中，依次输入所有程序 111，对所有程序 111 一次进行处理，但是在实施方式 6 的程序变换装置中，逐个

文件地输入程序，并逐个文件地进行程序变换处理。

图 62 是本发明实施方式 6 的程序变换装置 300 的结构图。程序变换装置 300 是接受用面向对象语言——C++语言描述的多个程序、输出可执行程序的装置，包括程序存储部 301、编译部 310、目标程序存储部 302、结合部 320、以及可执行程序存储部 303。

程序存储部 301 是逐个文件地存储作为程序变换装置 300 中的变换对象的、用面向对象语言——C++语言描述的多个程序 111a~111c 的存储装置。

编译部 310 是将多个程序 111a~111c 编译为多个目标程序 112a~112c 的处理部，由输入部 312、语法分析部 313、类继承关系分析部 314、虚函数重定义分析部 315、虚函数直接调用分析部 317、以及输出部 318 构成。

目标程序存储部 302 是存储与程序存储部 301 中保存着的程序 111a~111c 分别对应的目标程序 112a~112c 的存储装置。

结合部 320 是根据目标程序存储部 302 中存储着的多个目标程序 112a~112c 来生成 1 个可执行程序 324 的处理部，由输入部 321、虚函数调用变换部 322、以及链接部 323 构成。

可执行程序存储部 303 是存储从结合部 320 的链接部 323 输出的可执行程序 324 的存储装置。

结合以下说明的程序变换装置 300 执行的处理来说明其他处理部。

图 63 是程序变换装置 300 执行的处理的流程图。

输入部 312 选择程序存储部 301 中保存着的程序 111a~111c 中的 1 个文件(1 个程序)，交给语法分析部 313(S312)。其中，每当翻译部 310 结束对 1 个文件的编译处理时，输入部 312 就将下一个文件交给语法分析部 313。

语法分析部 313 分析从输入部 312 接受到的程序的语法，进行符号表的生成和语法树的生成等，将其分析结果交给类继承关系分析部 314(S313)。

类继承关系分析部 314 提取程序中包含的所有类并分析类间的继承关系，并将分析结果作为类的继承关系信息 113 保存到分析信息存储部 316 中(S314)。

虚函数重定义分析部 315 分析类的虚函数是否重定义了基类的虚函数，将分析结果作为虚函数重定义信息 114 保存到分析信息存储部 316 中(S315)。

虚函数直接调用分析部 317 分析程序中直接调用的虚函数及启动对象，并将分析结果作为虚函数直接调用信息 115 保存到分析信息存储部 316(S317)。

输出部 318 将虚函数调用分析部 317 的变换结果及分析信息存储部 316 中保存着的分析信息作为目标程序而保存到目标程序存储部 302 中(S318)。

输入部 321 接受目标程序存储部 302 中存储着的所有目标程序 112a~112c，交给虚函数调用变换部 322(S321)。

虚函数调用变换部 322 参照从输入部 321 接受到的目标程序 112a~112c 中包含的分析信息中保存的类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115，判定能否将虚函数调用变换为函数的直接调用，在能够变换的情况下，变换为函数的直接调用(S322)。

链接部 323 通过结合接虚函数调用变换部 322 变换后的所有目标程序来生成可执行程序 324，并将该可执行程序 324 记录到可执行程序存储部 303 中(S323)。

其中，语法分析处理(S313)、类继承关系分析处理(S314)、虚函数重定义分析处理(S315)、虚函数直接调用分析处理(S317)及虚函数调用变换处理(S322)的处理分别与实施方式 1 的语法分析处理(S103)、类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)及虚函数调用变换处理(S108)相同。

因此，与实施方式 1 的程序变换装置 100 同样，向程序变换装置 300 输入了图 13 所示的程序 111 的结果是，能够将虚函数调用的代码变换为

图 20 所示的代码。因此，能够提高执行可执行程序 324 时的执行速度。

其中，实施方式 2、实施方式 3、实施方式 4 及实施方式 5 中通过与实施方式 6 同样地构成，也能够得到与各实施方式同样的变换结果。

[实施方式 7]

在实施方式 1 的程序变换装置 100 中，同时进行了分析信息的收集和虚函数调用的变换，但是也可以分别执行分析信息的收集和虚函数调用的变换。在本实施方式的程序变换装置的特征在于，分别执行分析信息的收集和虚函数调用的变换。

图 64 是本发明实施方式 7 的程序变换装置 400 的结构图。程序变换装置 400 是接受用面向对象语言——C++语言描述的多个程序、输出目标程序的装置，包括程序存储部 401、预编译部 410、分析信息存储部 402、程序变换部 420、以及生成代码存储部 403。

程序存储部 401 是逐个文件地存储程序变换装置 400 中作为变换对象的、用面向对象语言——C++语言描述的多个程序 111a~111c 的存储装置。

预编译部 410 是依次接受程序存储部 401 中保存着的所有程序 111a~111c、生成各种分析信息(类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115)的处理部，由输入部 412、语法分析部 413、类继承关系分析部 414、虚函数重定义分析部 415、以及虚函数直接调用分析部 416 构成。

程序变换部 420 是参照分析信息存储部 402 中保存着的类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115，来判定能否将程序 111a~111c 中的虚函数调用变换为函数的直接调用，并在能够变换的情况下，变换为函数的直接调用的处理部。该程序变换部 420 由输入部 421、语法分析部 422、虚函数调用变换部 423、以及输出部 424 构成。

生成代码存储部 403 是存储从程序变换部 420 输出的目标程序 112 的存储装置。

结合以下说明的程序变换装置 400 执行的处理来说明其他处理部。

图 65 是程序变换装置 400 执行的处理的流程图。

输入部 412 依次输入程序存储部 401 中保存着的所有程序 111a~111c，交给语法分析部 413(S412)。

语法分析部 413 分析从输入部 412 接受到的程序 111a~111c 的语法，进行符号表的生成或语法树的生成等，并将其分析结果交给类继承关系分析部 414(S413)。

类继承关系分析部 414 提取程序 111a~111c 中包含的所有类并分析类间的继承关系，将分析结果作为类的继承关系信息 113 保存到分析信息存储部 402 中(S414)。

虚函数重定义分析部 415 分析类的虚函数是否重定义了基类的虚函数，并将分析结果作为虚函数重定义信息 114 而保存到分析信息存储部 402 中(S415)。

虚函数直接调用分析部 416 分析程序 111a~111c 中直接调用的虚函数及启动对象，并将分析结果作为虚函数直接调用信息 115 保存到分析信息存储部 402 中(S416)。

输入部 421 依次输入分析信息存储部 402 中保存着的类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115、以及程序存储部 401 中保存着的程序 111a~111c，交给语法分析部 422(S421)。

语法分析部 422 分析从输入部 421 接受到的程序 111a~111c 的语法，进行符号表的生成或语法树的生成等，将其分析结果交给虚函数调用变换部 423(S422)。此外，语法分析部 422 将类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115 交给虚函数调用变换部 423。

虚函数调用变换部 423 参照从语法分析部 422 接受到的类的继承关系信息 113、虚函数重定义信息 114 及虚函数直接调用信息 115，判定能否将虚函数调用变换为函数的直接调用，在能够变换的情况下，变换为函数的直接调用(S423)。

输出部 424 将通过各步骤变换的程序作为目标程序 112 记录到生成代码存储部 403 中。

其中，语法分析处理(S413、S422)、类继承关系分析处理(S414)、虚函数重定义分析处理(S415)、虚函数直接调用分析处理(S416)及虚函数调用变换处理(S423)分别与实施方式 1 的语法分析处理(S103)、类继承关系分析处理(S104)、虚函数重定义分析处理(S105)、虚函数直接调用分析处理(S107)及虚函数调用变换处理(S108)相同。

因此，与实施方式 1 的程序变换装置 100 同样，向程序变换装置 400 输入了图 13 所示的程序 111 的结果是，能够将虚函数调用的代码变换为图 20 所示的代码。因此，能够提高执行目标程序 112 时的执行速度。

其中，实施方式 2、实施方式 3、实施方式 4 及实施方式 5 通过构成为与实施方式 7 同样，也能够得到与各实施方式同样的变换结果。

产业上的可利用性

本发明能够适用于编译器，特别是能够适用于变换用面向对象语言描述的程序的编译器等。

```
class A {  
    virtual void f(int)  
    virtual void g(int)  
    virtual void h(int)  
}  
class B : public A {  
    void g(int)  
}  
B obj_b;  
void t1(void)  
{  
    t2(&obj_b)  
}  
void t2(A* p)  
{  
    p->g(10); // 虚函数调用  
}
```

图 1

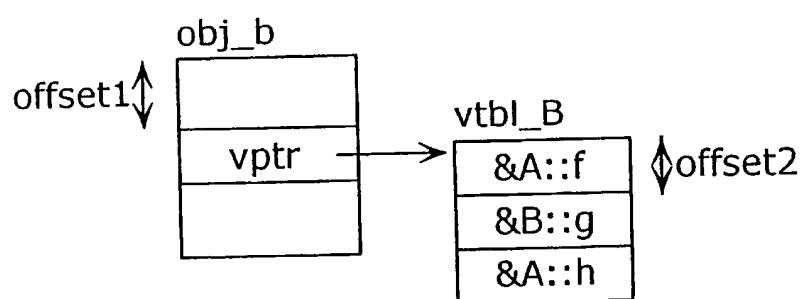


图2

虚函数调用

```
mov (R0,offset1), R2  
mov (R2,offset2), R2  
mov 10, R1  
call (R2)
```

图3

函数的直接调用

```
mov 10, R1  
call _g_A
```

图4

```

class A {
    virtual void f(void)
    virtual void g(void)
}
class B : public A {
    void f(void)
}
class C : public B {
    void g(void)
}
class D : public B {
    void f(void)
    void g(void)
}
void h(A* p) ← (A)
{
    p->f()
}
void A::f(void) ← (B)
{
    this->g()
}

```

图5

```

void h(A* p)
{
    取得虚函数的地址
    if(虚函数的地址是A::f)
    {
        p->A::f(); // A::f 的直接调用
    } else {
        p->f(); // 虚函数调用
    }
}

```

图6

· 没有A的派生类时

```
void h(A* p)
{
    p->A::f(); // A::f 的直接调用
}
```

· 有A的派生类时

```
void h(A* p)
{
    p->f(); // 虚函数调用
}
```

图 7

```
void A::f(void)
{
    取得虚函数的地址
    if(虚函数的地址是A::g)
    {
        this->A::g(); // A::g 的直接调用
    } else {
        this->g(); // 虚函数调用
    }
}
```

图 8

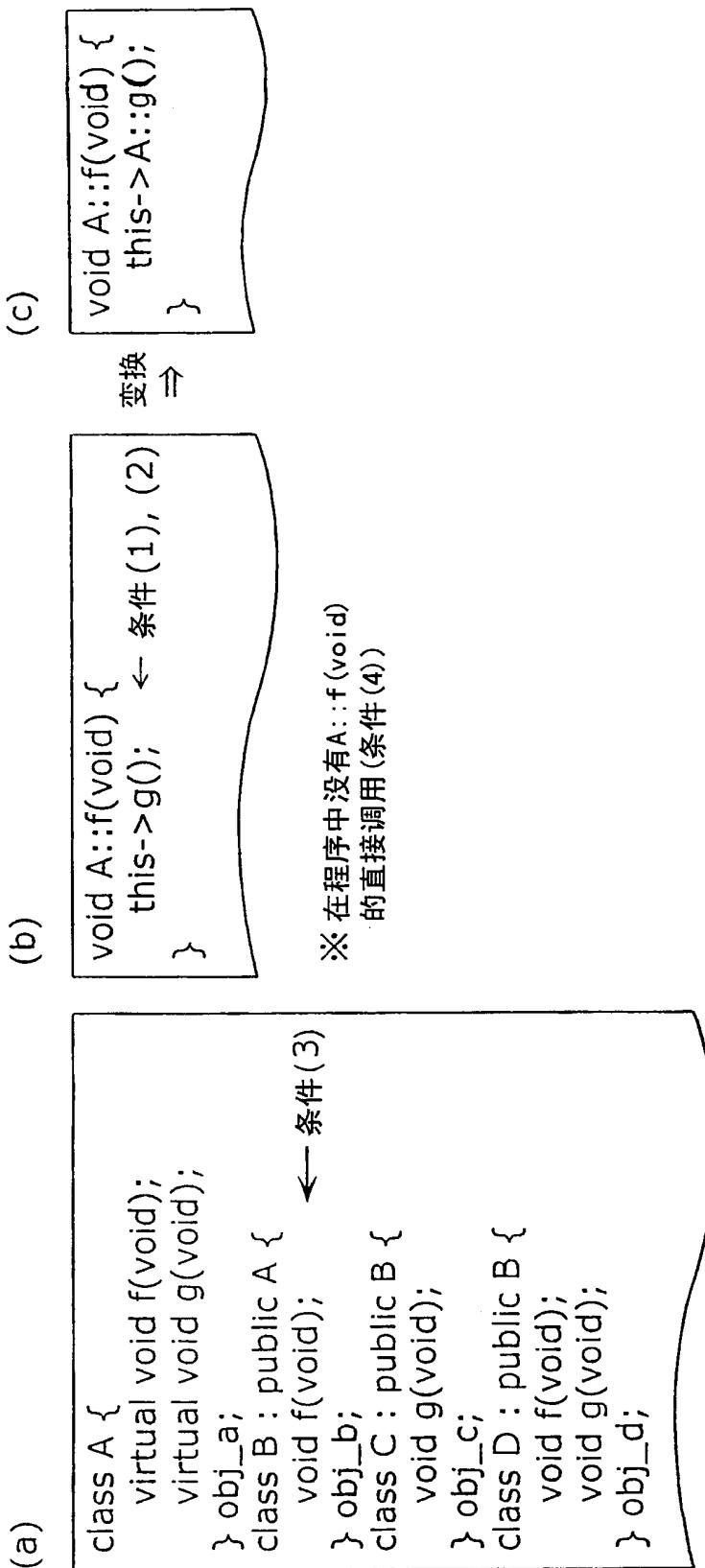


图9

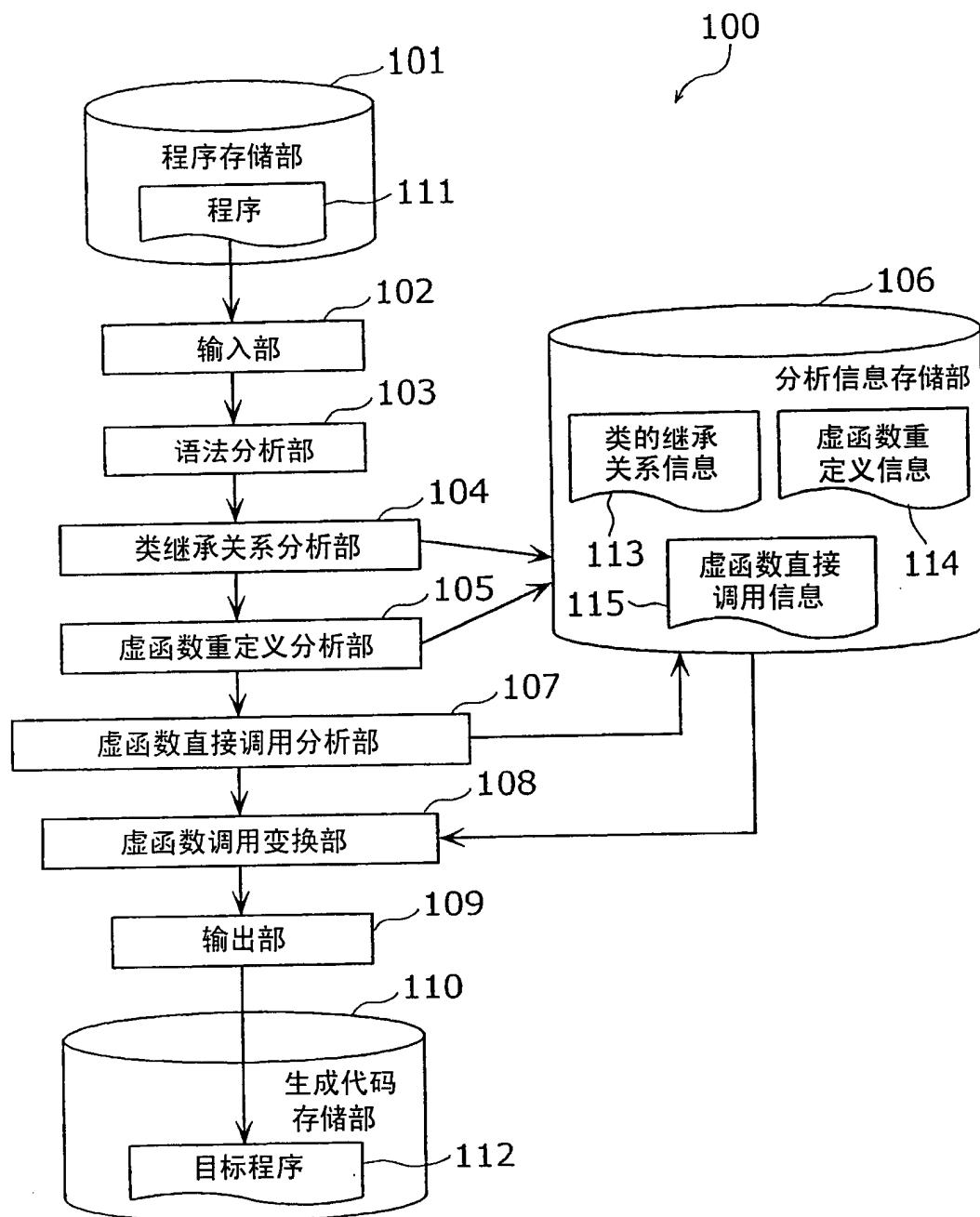


图10

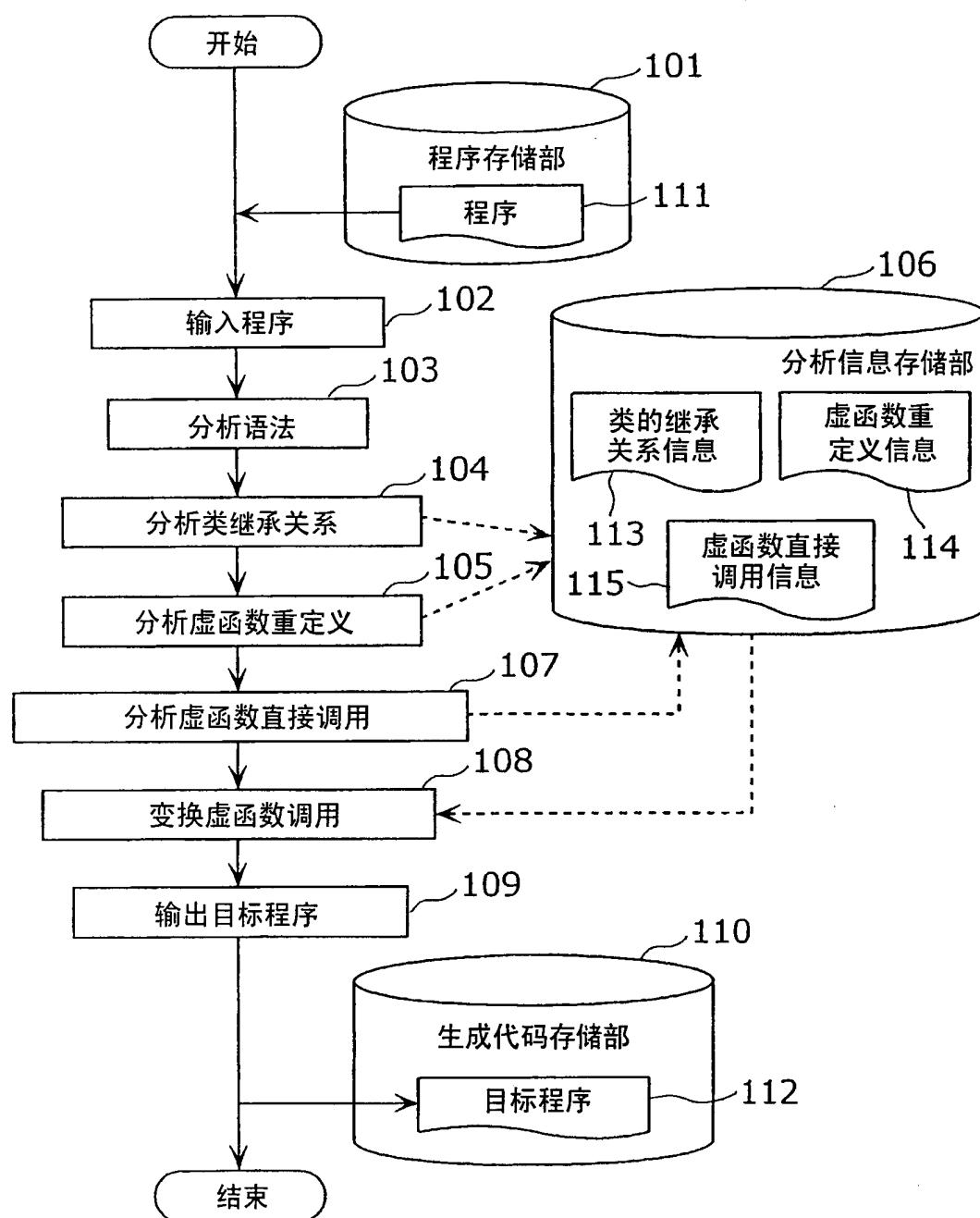


图 11

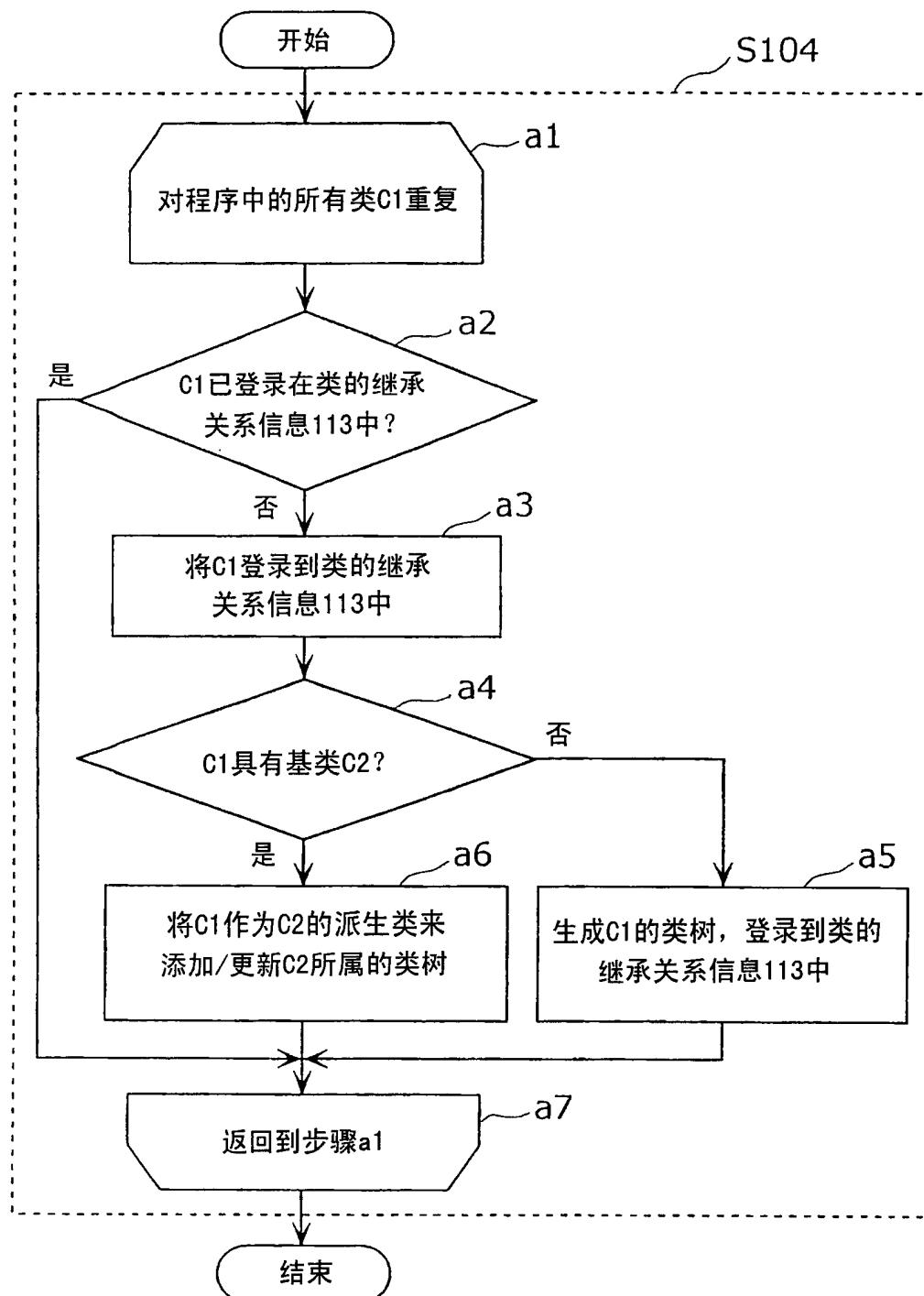


图12

111

```
class A {
    virtual void f(void)
    virtual void g(void)
    virtual void h(void)
}
class B : public A {
    void f(void)
    void g(void)
}
class C : public B {

}
class D : public B {
    void g(void)
    void h(void)
}
void A::f(void)
{
    this->g()
}
void B::g(void)
{
    this->h()
}
void t(void)
{
    C obj_c
    obj_c.g()
}
```

图 13

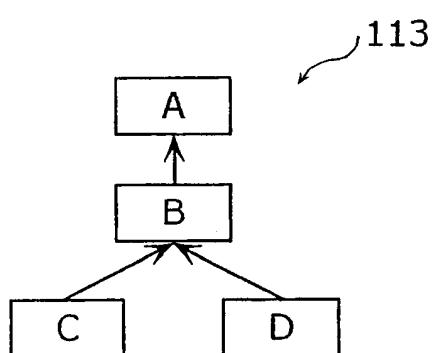


图14

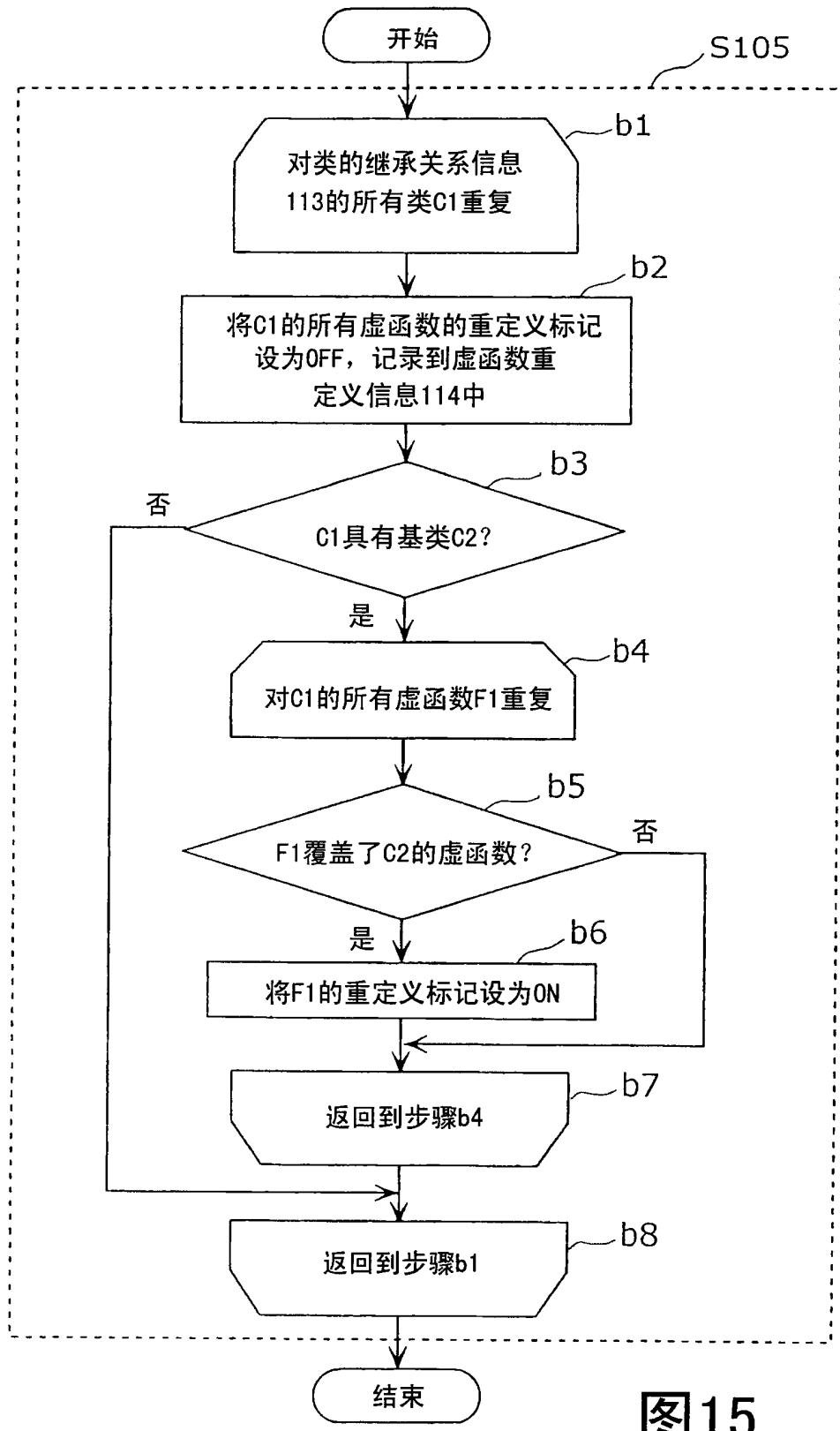


图15

114

A	f(void)	OFF
A	g(void)	OFF
A	h(void)	OFF
B	f(void)	ON
B	g(void)	ON
B	h(void)	OFF
C	f(void)	OFF
C	g(void)	OFF
C	h(void)	OFF
D	f(void)	OFF
D	g(void)	ON
D	h(void)	ON

图16

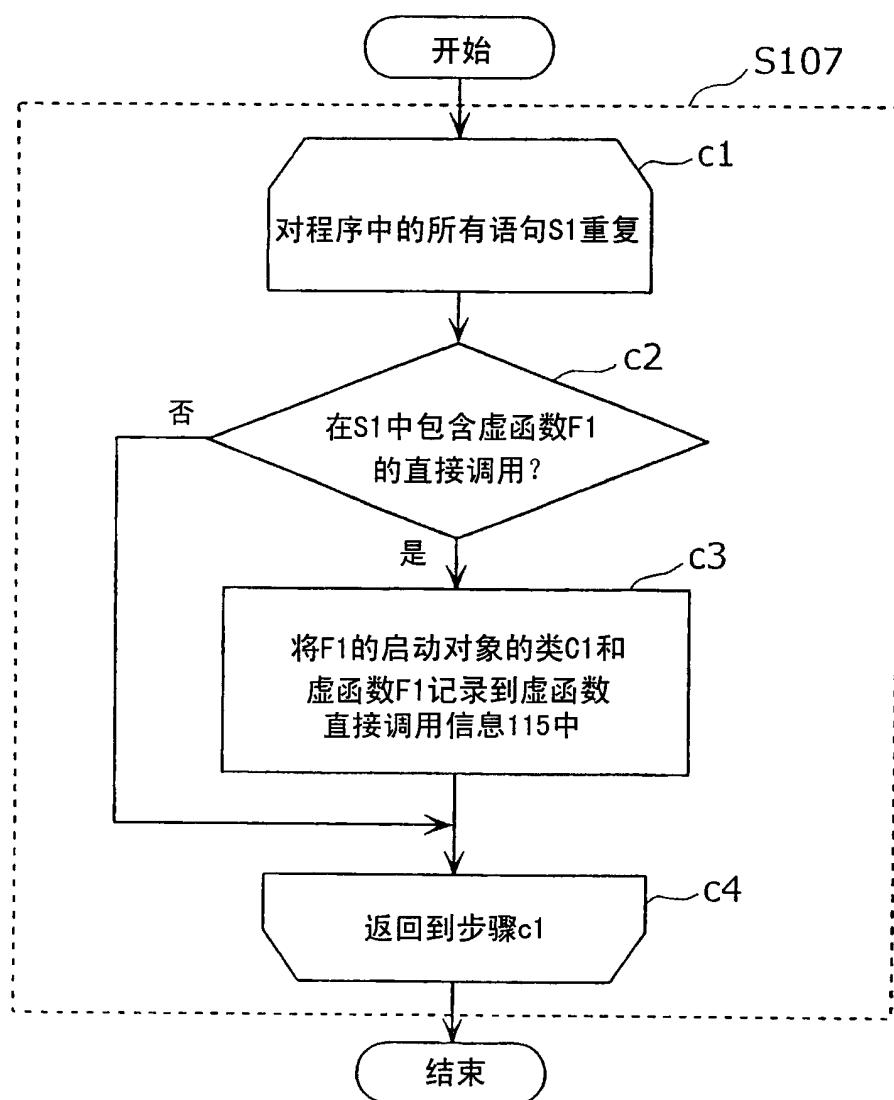


图 17

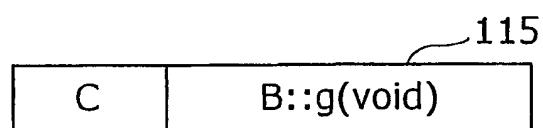


图18

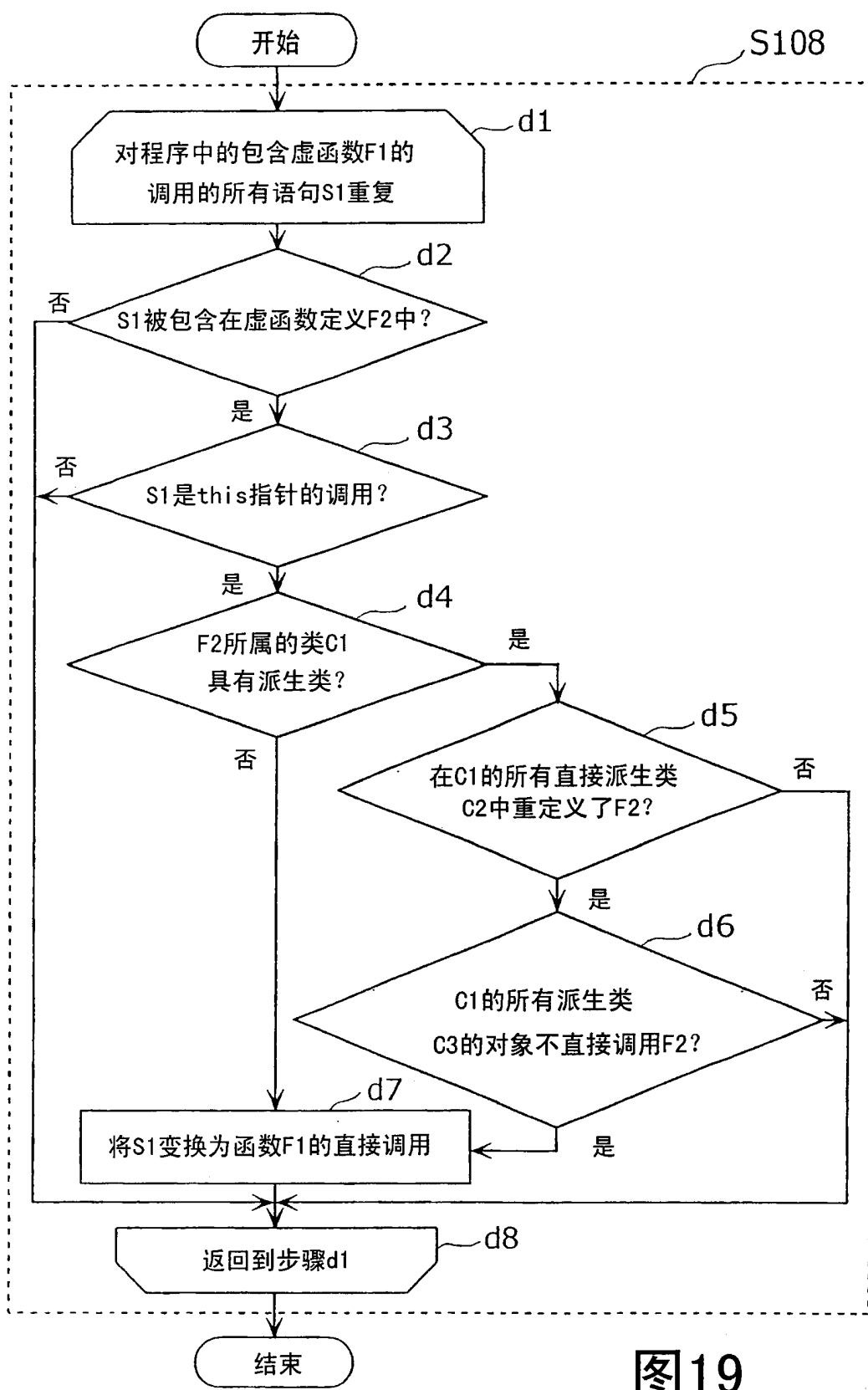


图 19

```
void A::f(void)
{
    this->A::g();      // A::g 的直接调用
}
```

图20

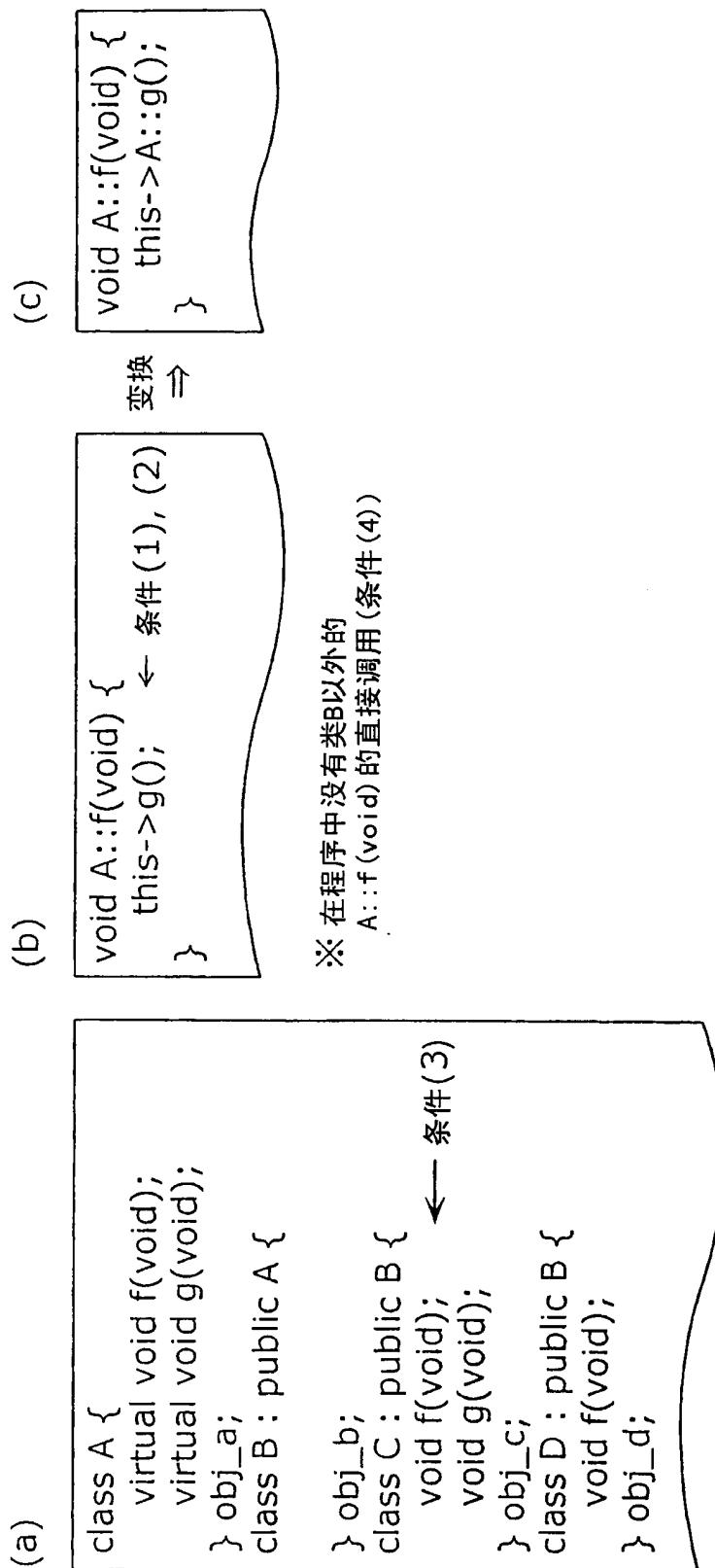


图 21

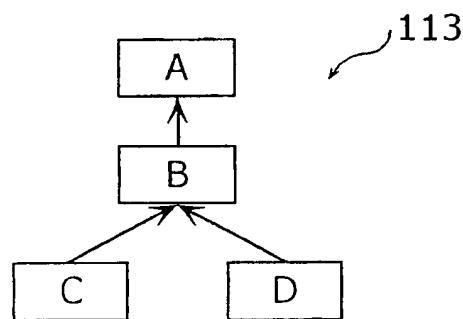


图22

114

A	f(void)	OFF
A	g(void)	OFF
A	h(void)	OFF
B	f(void)	ON
B	g(void)	ON
B	h(void)	OFF
C	f(void)	OFF
C	g(void)	OFF
C	h(void)	OFF
D	f(void)	OFF
D	g(void)	ON
D	h(void)	ON

图23

115

C	B::g(void)
---	------------

图24

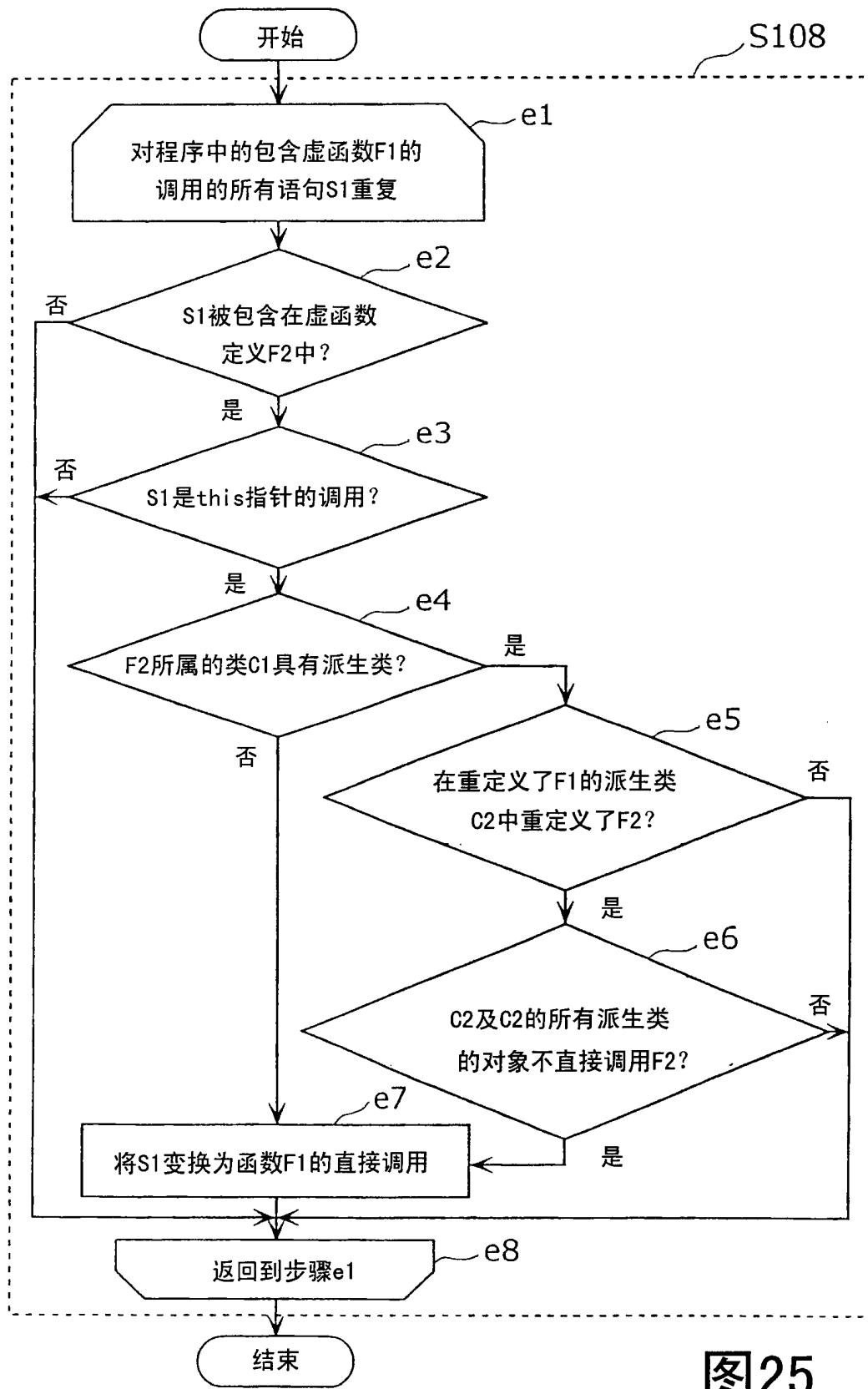


图25

```
void A::f(void)
{
    this->A::g();      // A::g 的直接调用
}
void B::g(void)
{
    this->B::h();      // B::h 的直接调用
}
```

图26

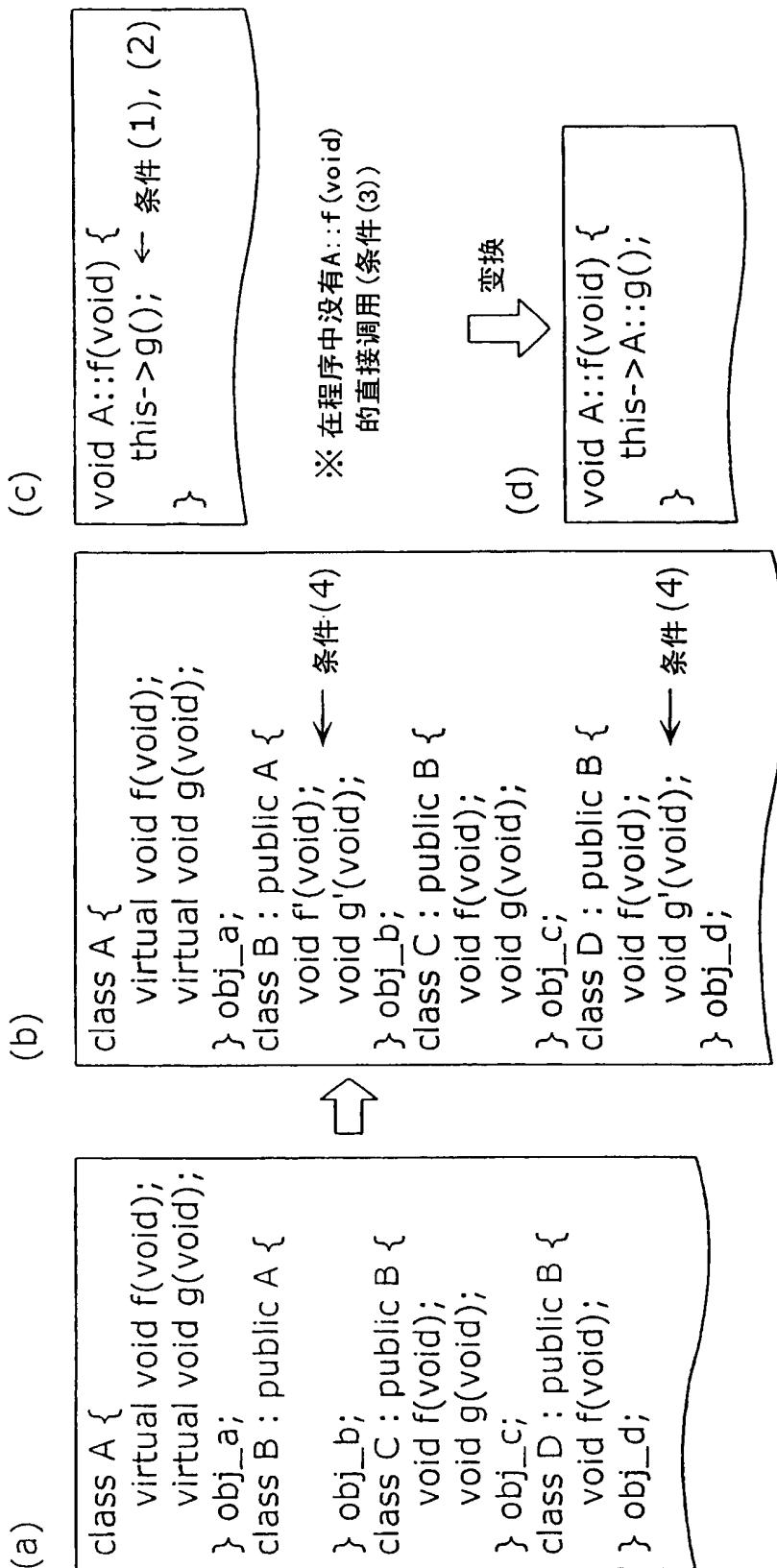


图27

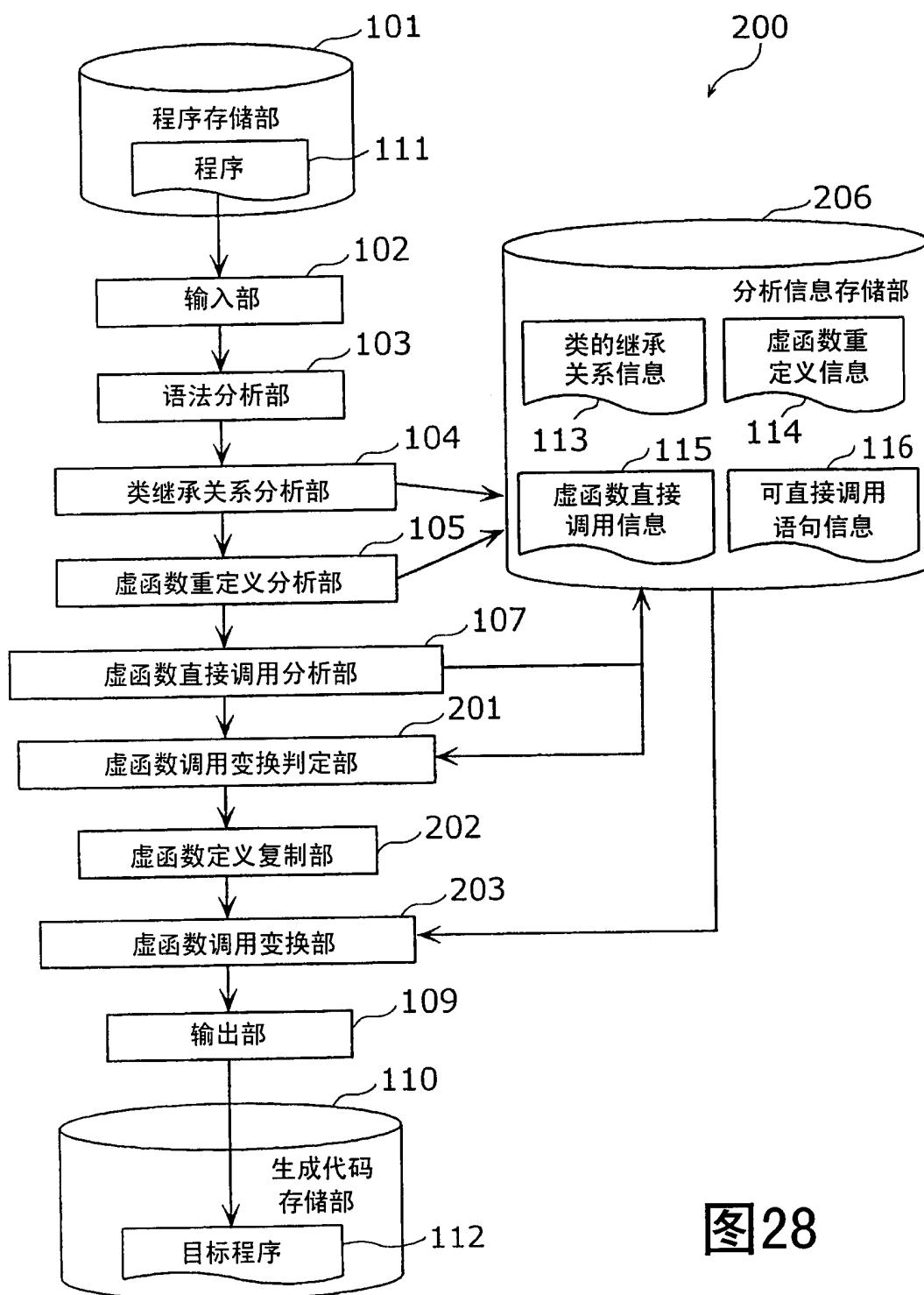


图28

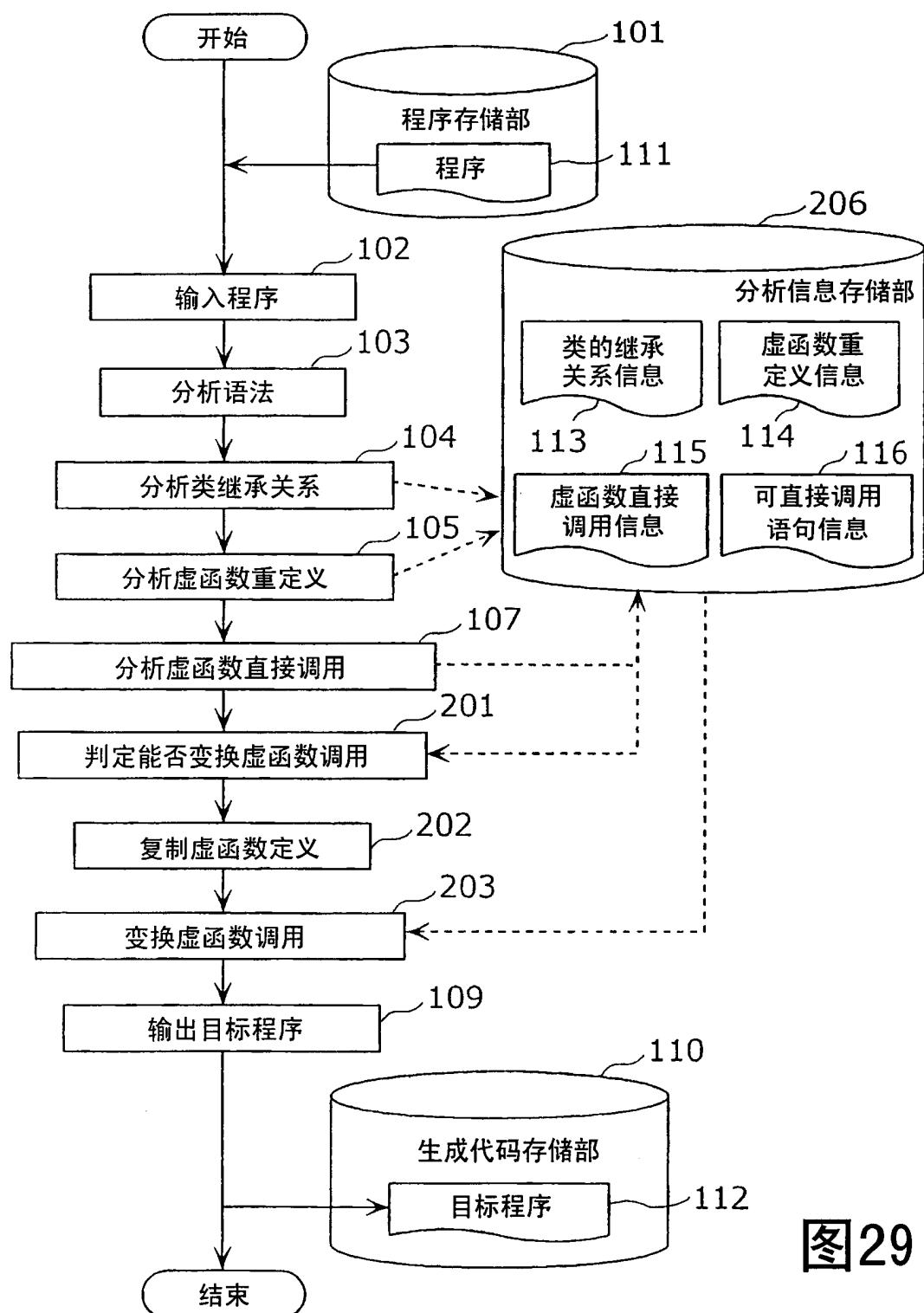


图29

111

```
class A {
    virtual void f(void)
    virtual void g(void)
    virtual void h(void)
}
class B : public A {
    void g(void)
}
class C : public B {

}
class D : public B {
    void h(void)
}
void A::f(void)
{
    this->g()
}
void B::g(void)
{
    this->h()
}
void t(void)
{
    C obj_c
    obj_c.g()
}
```

图30

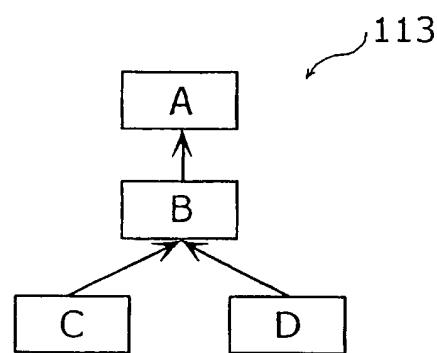


图31

114

A	f(void)	OFF
A	g(void)	OFF
A	h(void)	OFF
B	f(void)	OFF
B	g(void)	ON
B	h(void)	OFF
C	f(void)	OFF
C	g(void)	OFF
C	h(void)	OFF
D	f(void)	OFF
D	g(void)	OFF
D	h(void)	ON

图32

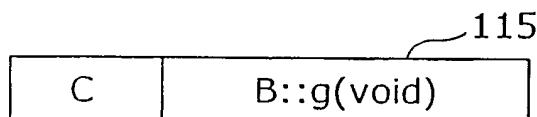


图33

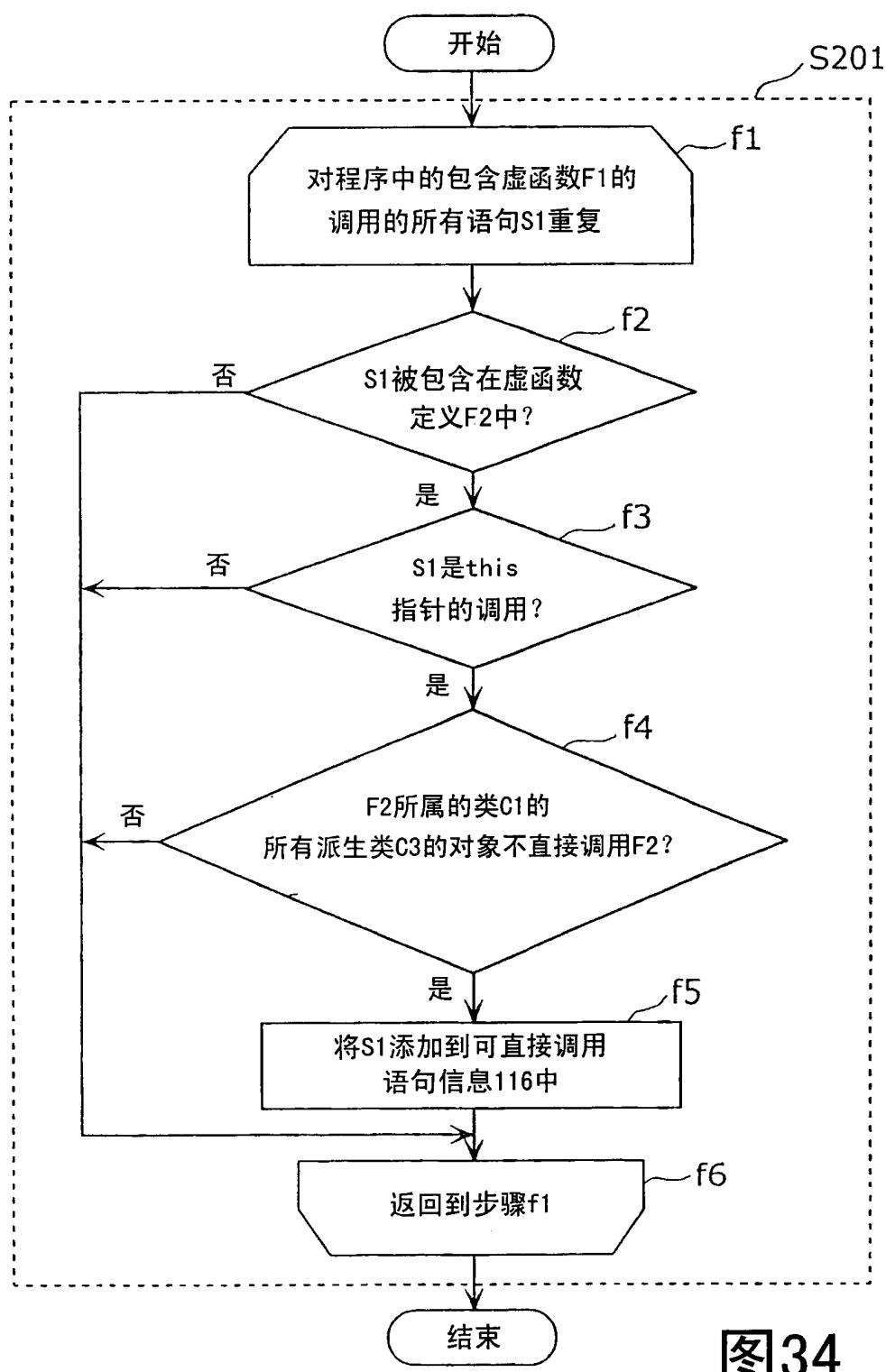


图 34

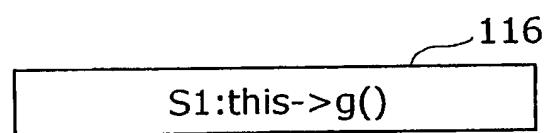
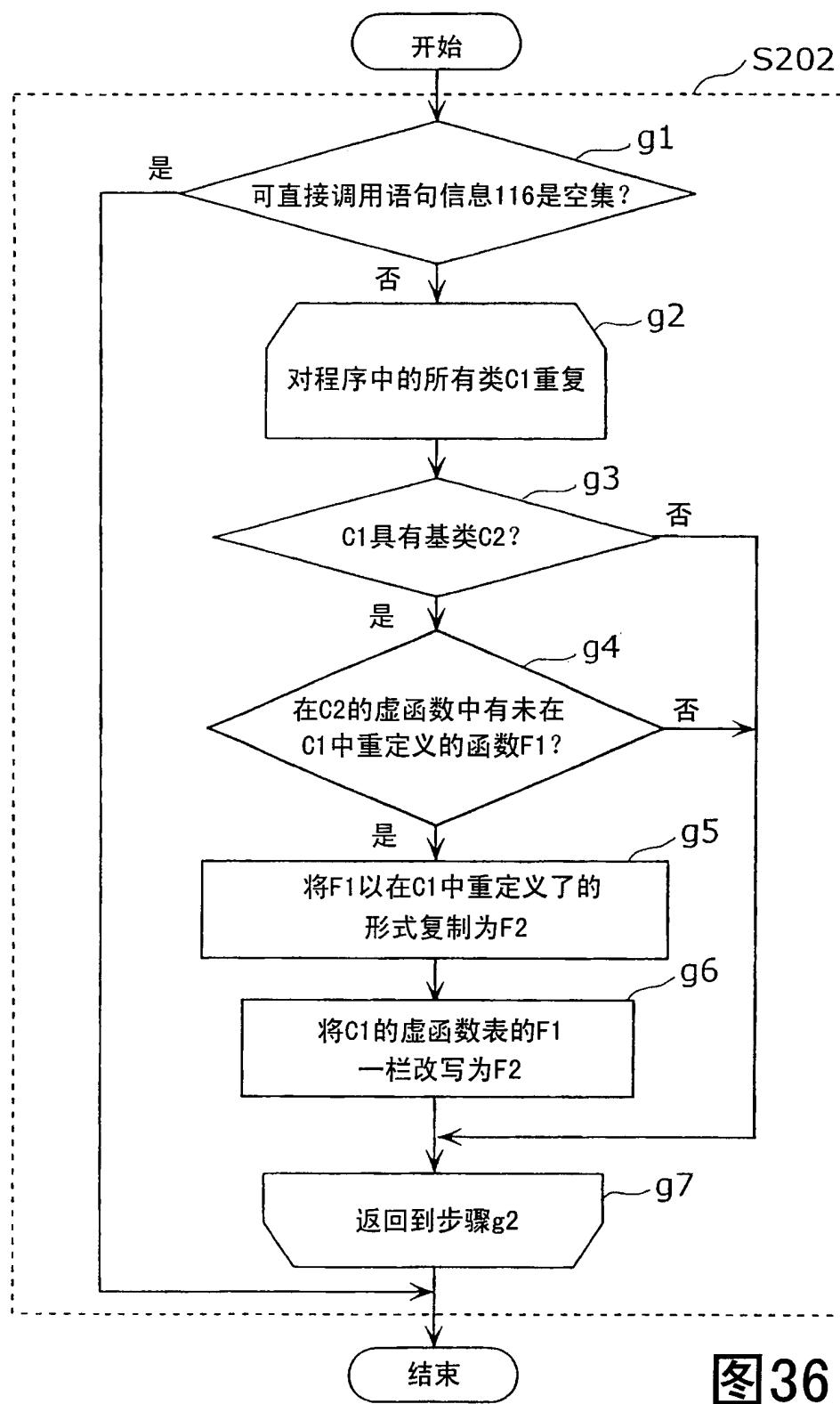


图35



冬 36

```
class A {  
    virtual void f(void)  
    virtual void g(void)  
    virtual void h(void)  
}  
class B : public A {  
    void f'(void)  
    void g(void)  
    void h'(void)  
}  
class C : public B {  
    void f'(void)  
    void g'(void)  
    void h'(void)  
}  
class D : public B {  
    void f'(void)  
    void g'(void)  
    void h(void)  
}
```

图37

vtbl_A

A::f(void)
A::g(void)
A::h(void)

vtbl_B

B::f'(void)
B::g(void)
B::h'(void)

vtbl_C

C::f'(void)
C::g'(void)
C::h'(void)

vtbl_D

D::f'(void)
D::g'(void)
D::h(void)

图 38

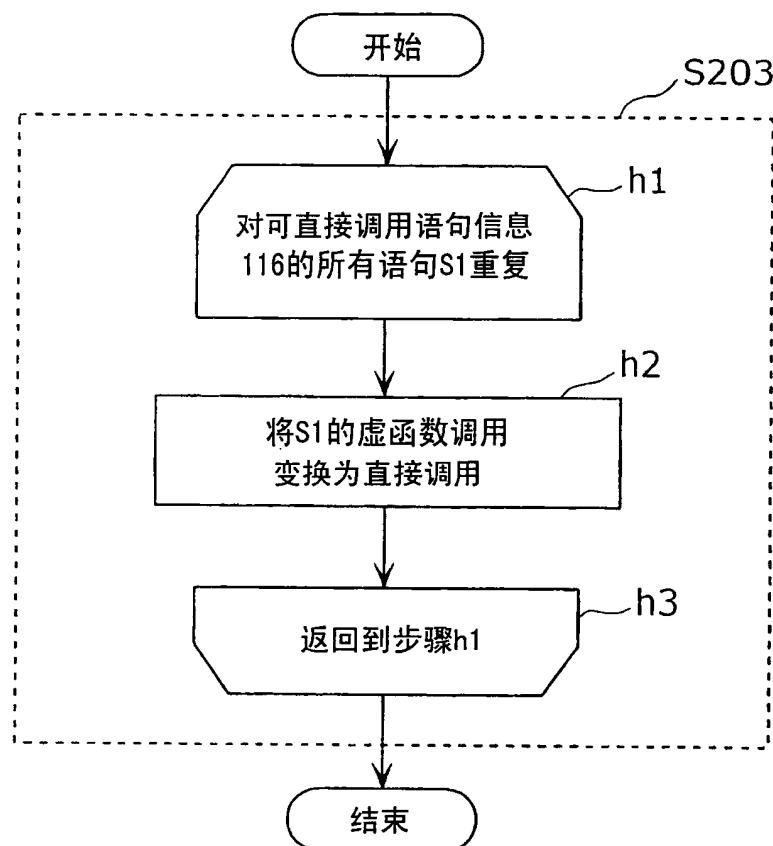


图39

```
void A::f(void)
{
    this->A::g(); // A::g 的直接调用
}
```

图40

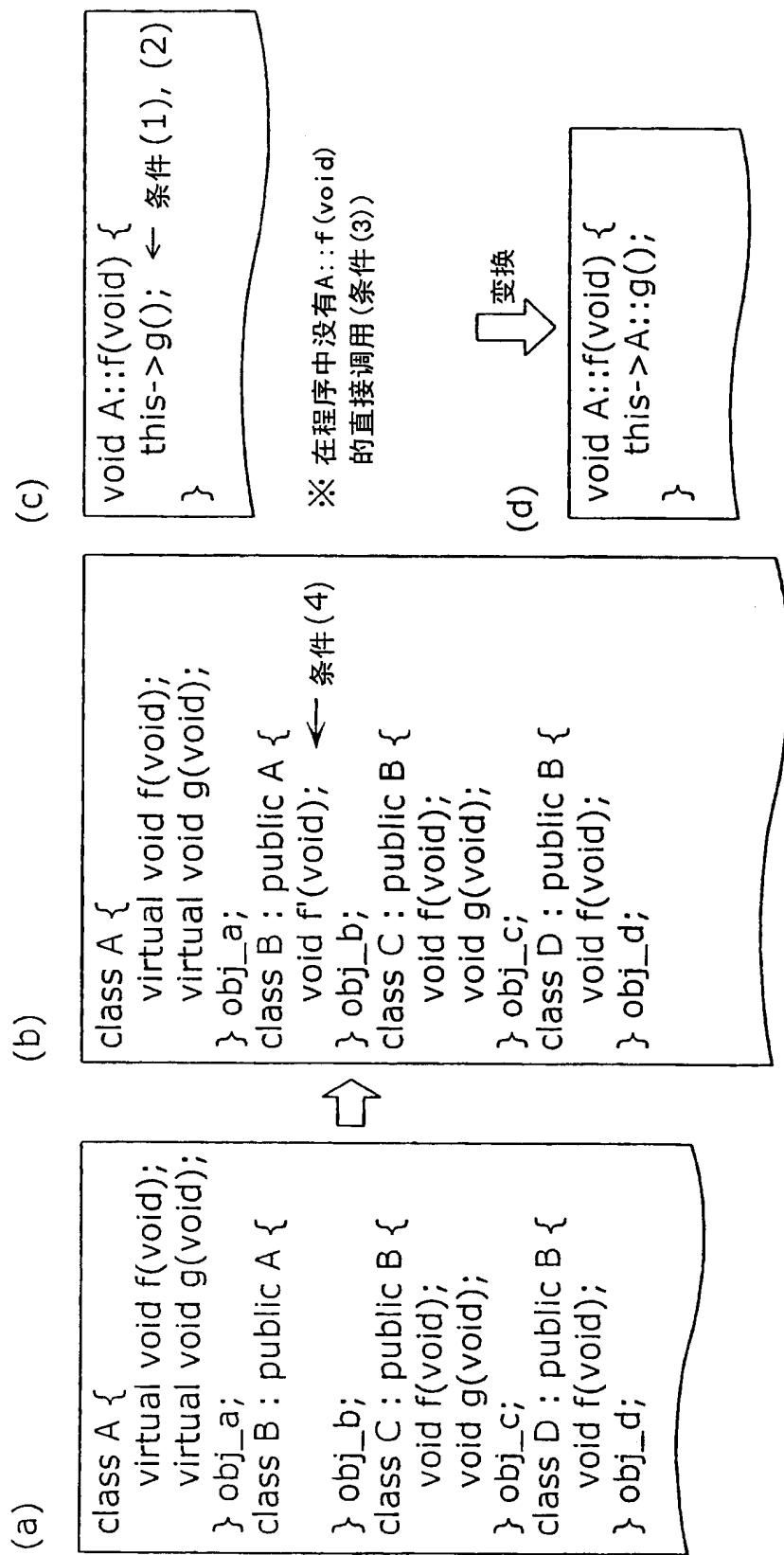


图41

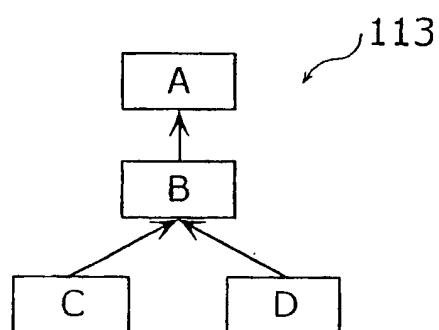


图42

114

A	f(void)	OFF
A	g(void)	OFF
A	h(void)	OFF
B	f(void)	OFF
B	g(void)	ON
B	h(void)	OFF
C	f(void)	OFF
C	g(void)	OFF
C	h(void)	OFF
D	f(void)	OFF
D	g(void)	OFF
D	h(void)	ON

图43

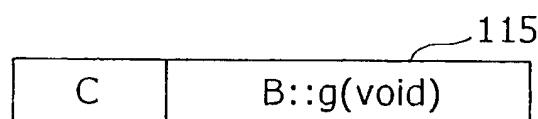


图44

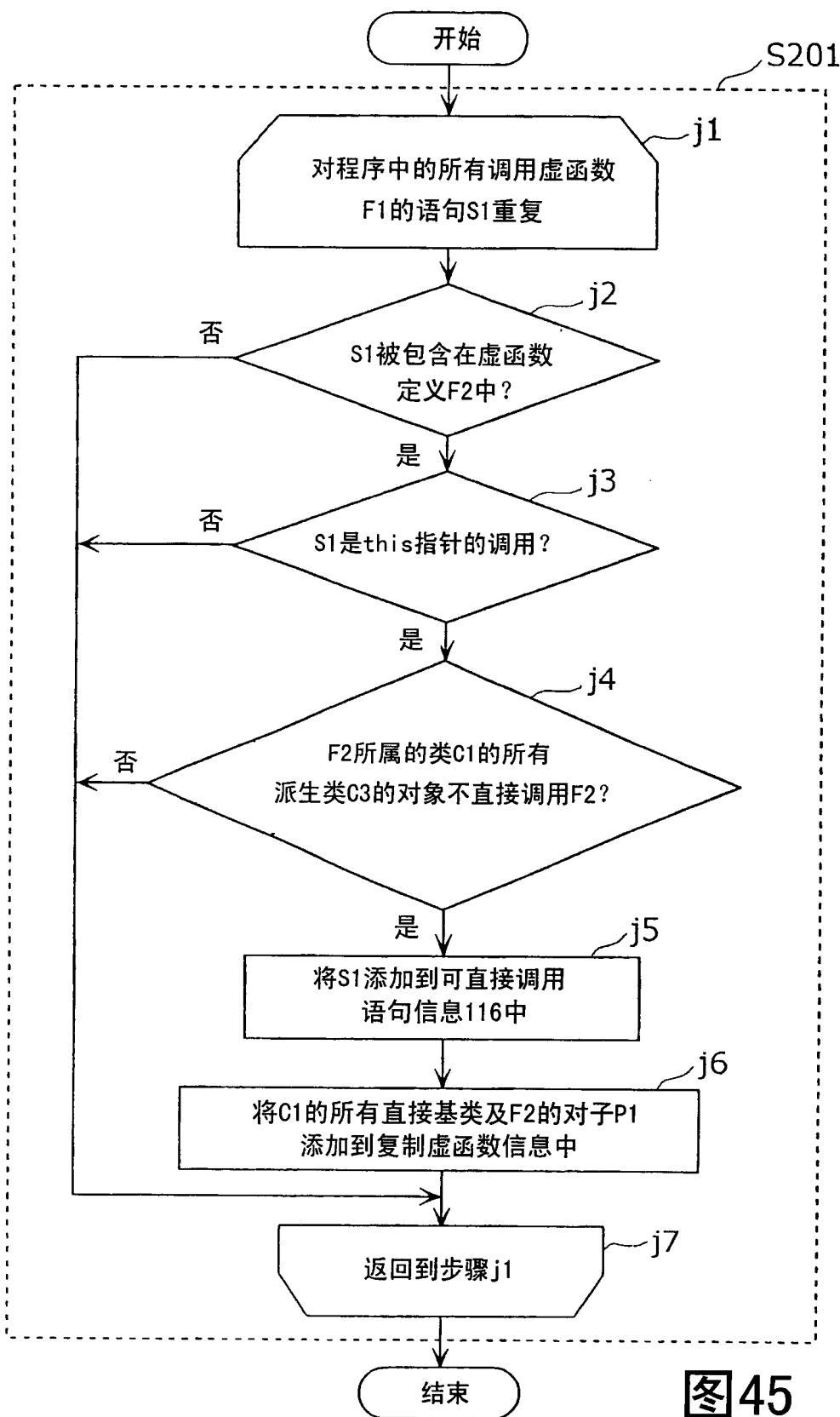


图45

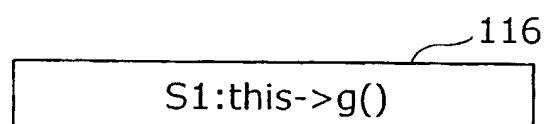


图46

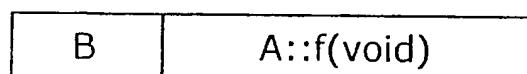


图47

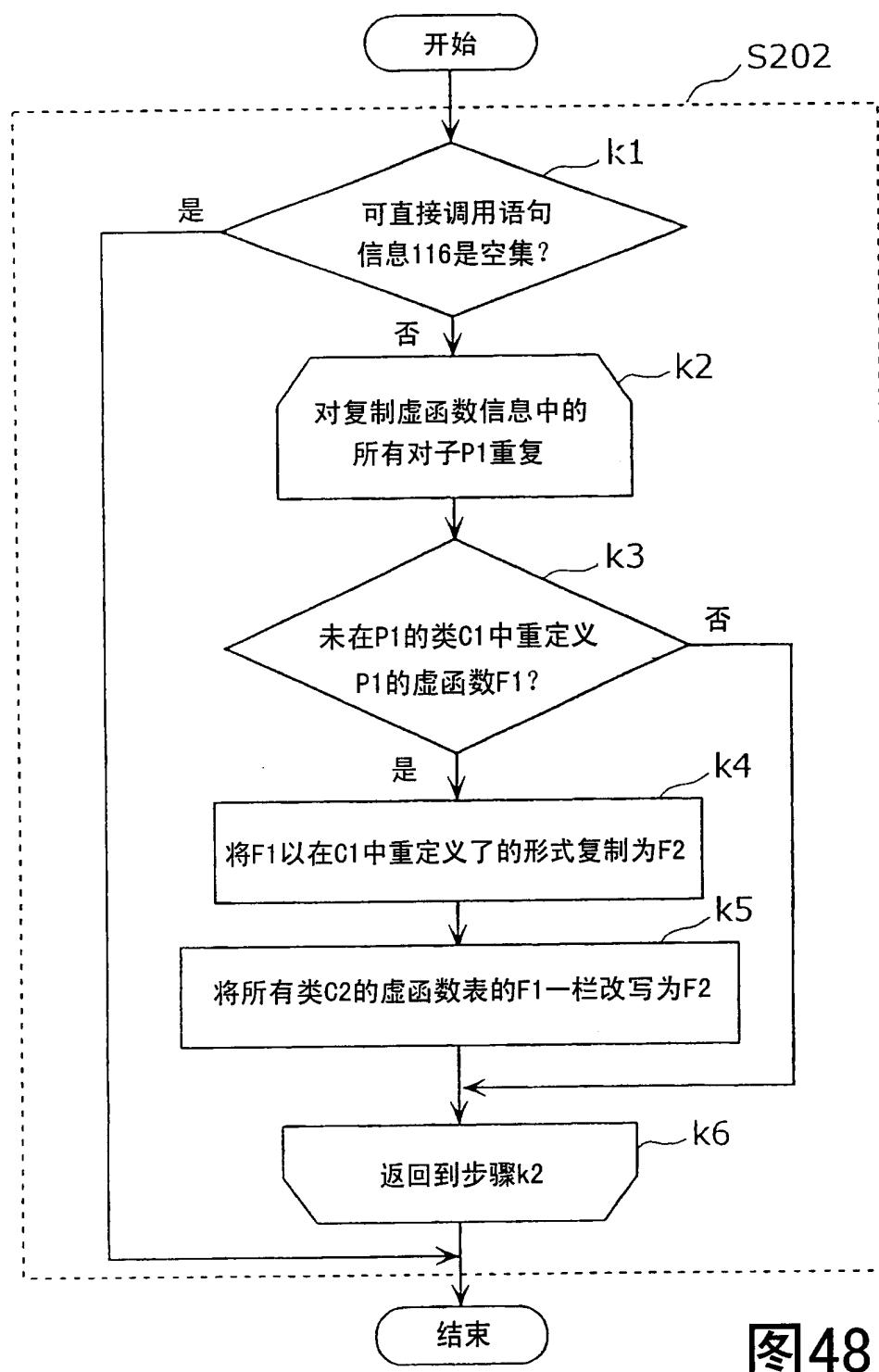


图 48

```
class A {  
    virtual void f(void)  
    virtual void g(void)  
    virtual void h(void)  
}  
class B : public A {  
    void f'(void)  
    void g(void)  
}  
class C : public B {  
}  
class D : public B {  
    void h(void)  
}
```

图49

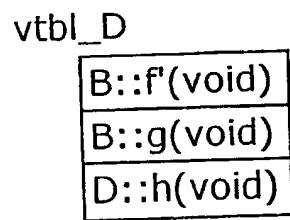
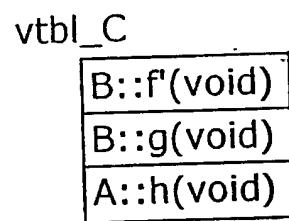
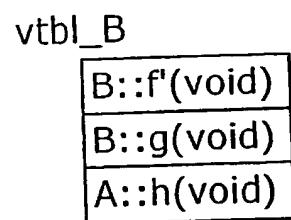
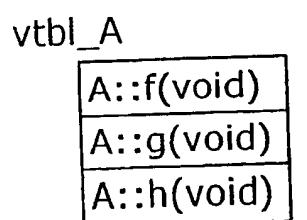


图50

```
void A::f(void)
{
    this->A::g();      // A::g 的直接调用
}
```

图51

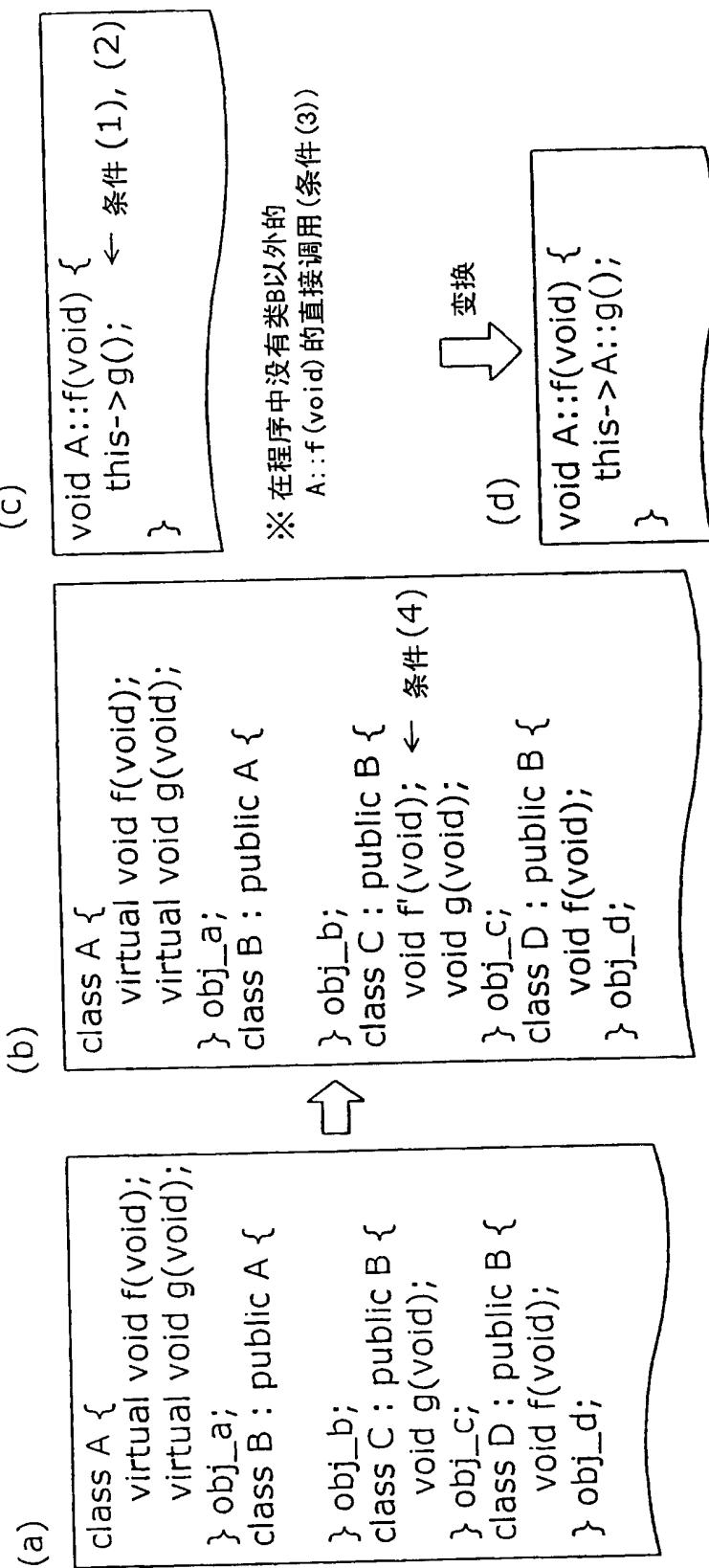


图52

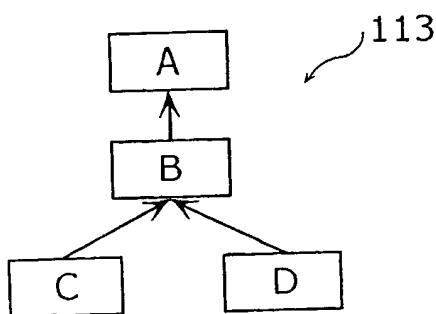


图53

114

A	f(void)	OFF
A	g(void)	OFF
A	h(void)	OFF
B	f(void)	OFF
B	g(void)	ON
B	h(void)	OFF
C	f(void)	OFF
C	g(void)	OFF
C	h(void)	OFF
D	f(void)	OFF
D	g(void)	OFF
D	h(void)	ON

图54

115

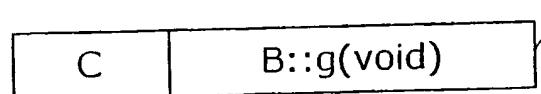


图55

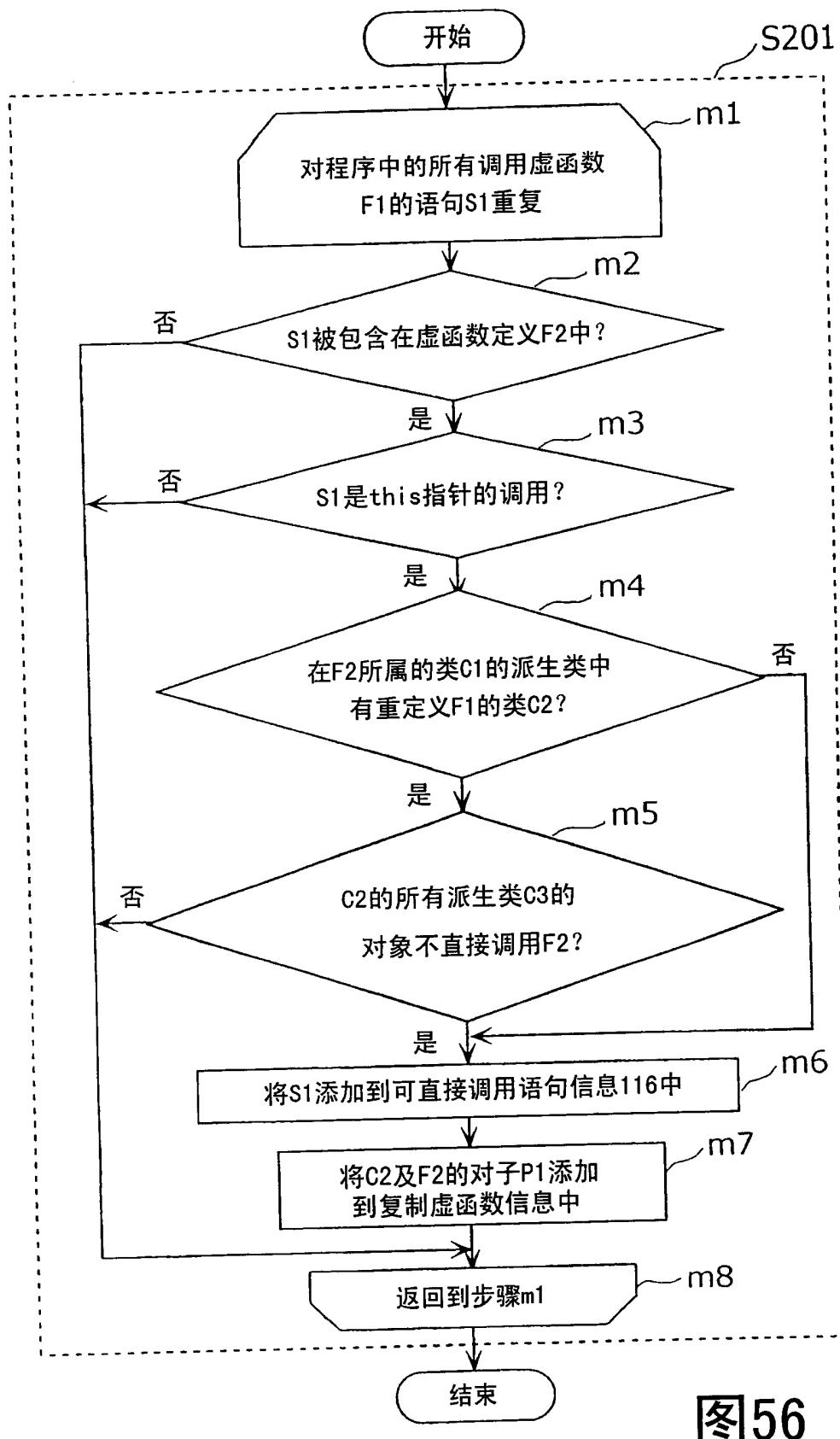


图56

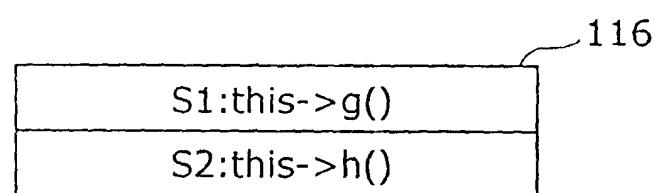


图57

B	A::f(void)
D	B::g(void)

图58

```
class A {  
    virtual void f(void)  
    virtual void g(void)  
    virtual void h(void)  
}  
class B : public A {  
    void f'(void)  
    void g(void)  
}  
class C : public B {  
}  
class D : public B {  
    void g'(void)  
    void h(void)  
}
```

图59

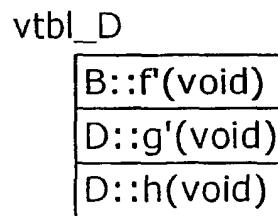
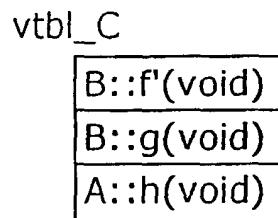
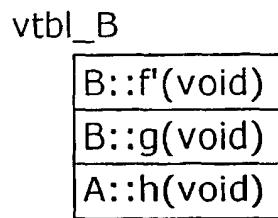
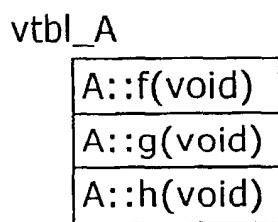


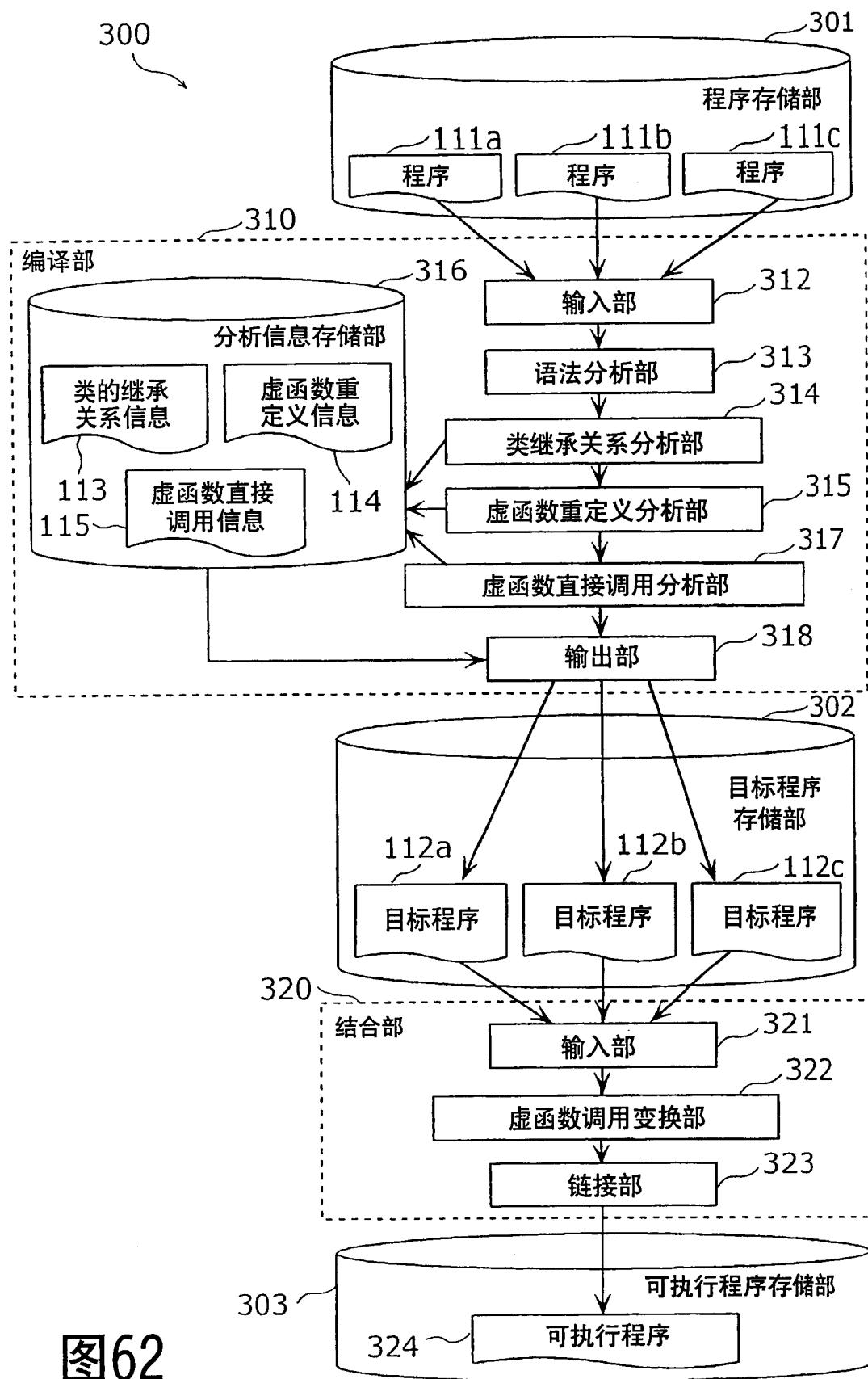
图60

```

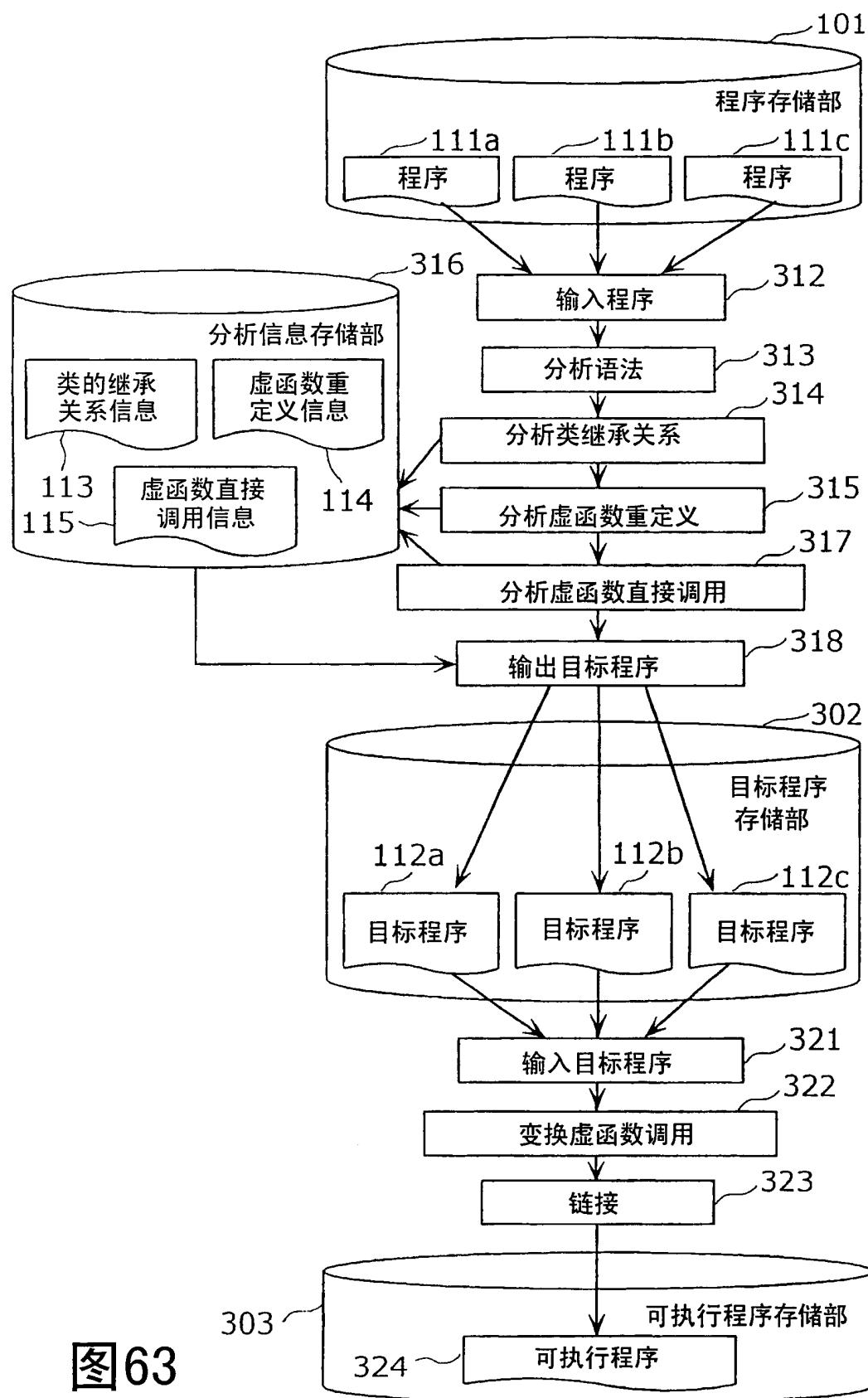
void A::f(void)
{
    this->A::g();      // A::g 的直接调用
}
void B::g(void)
{
    this->B::h();      // B::h 的直接调用
}

```

图61



冬 62



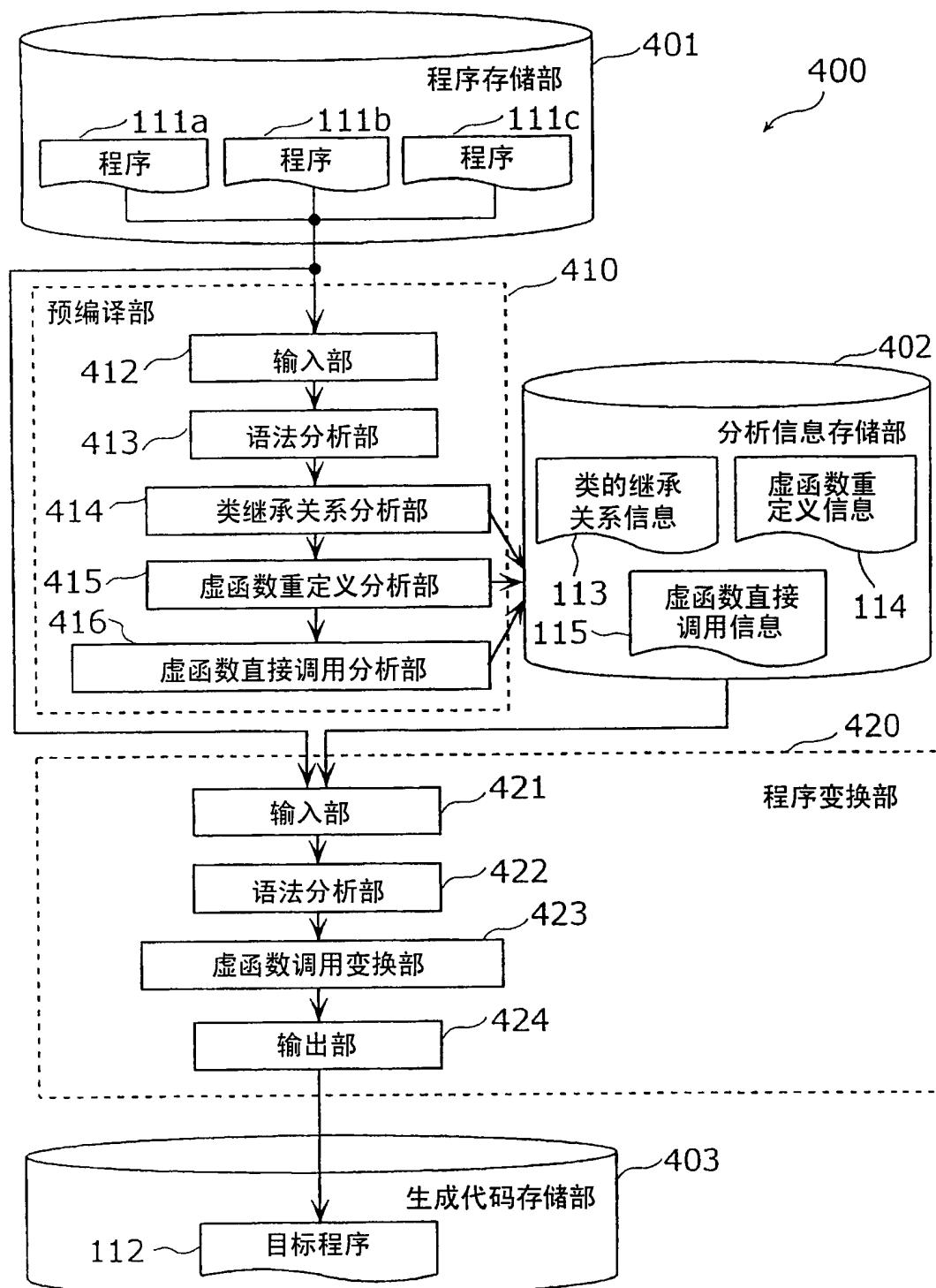


图64

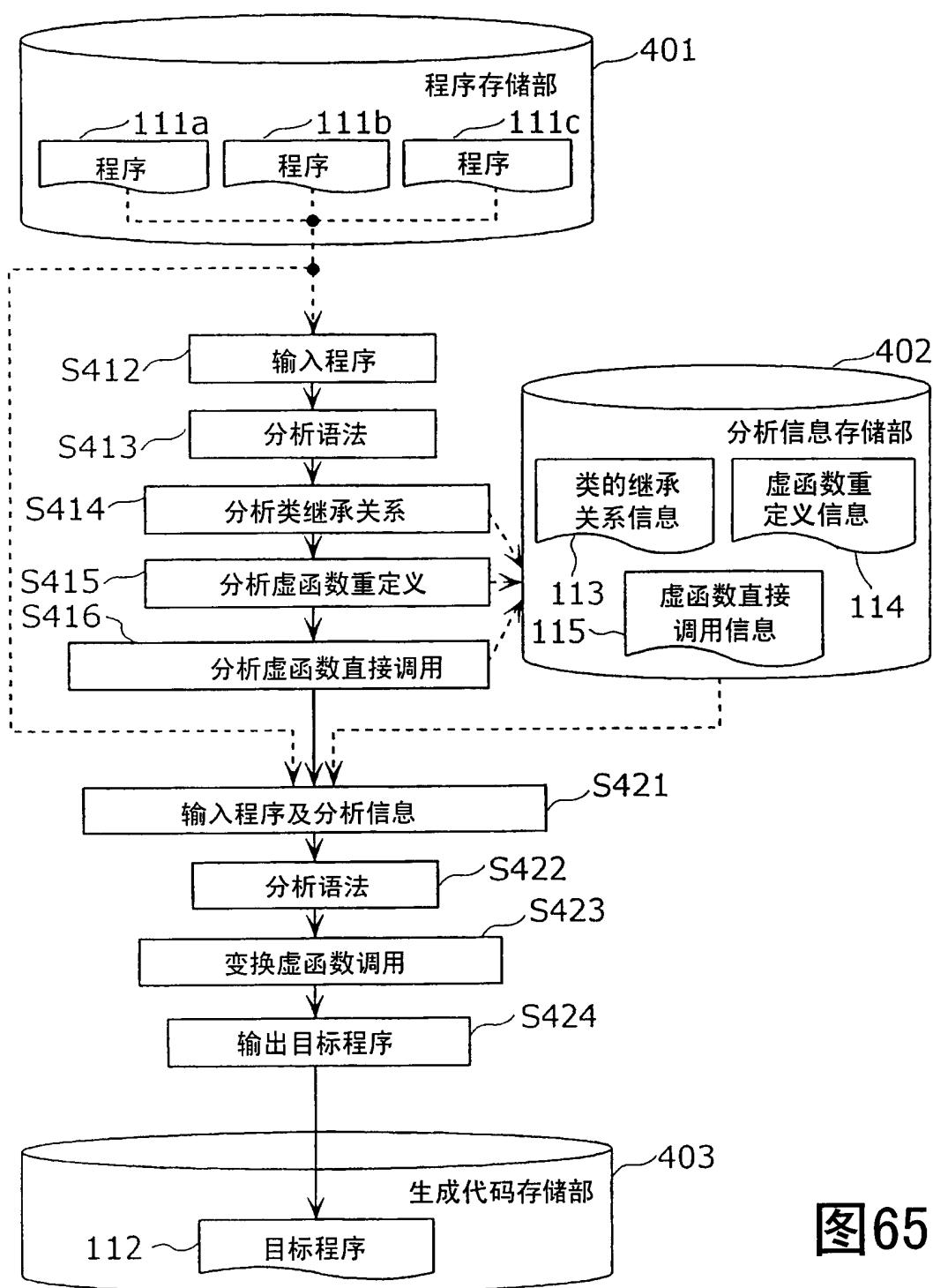


图65