[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 17/60

[12] 发明专利申请公开说明书

[21] 申请号 01820158.X

[43] 公开日 2004 年 3 月 3 日

[11] 公开号 CN 1479904A

[22] 申请日 2001.11.30 [21] 申请号 01820158.X
[30] 优先权
[32] 2000.12.8 [33] US [31] 60/251,862
[86] 国际申请 PCT/DK01/00801 2001.11.30
[87] 国际公布 WO02/46980 英 2002.6.13
[85] 进入国家阶段日期 2003.6.6
[71] 申请人 产品配置软件股份有限公司
地址 丹麦哥本哈根
[72] 发明人 雅各布·利希腾贝格
亨里克·雷尔夫·安德森
亨里克·许尔加德 叶斯佩尔·穆勒
安诺斯·斯腾·拉斯穆森

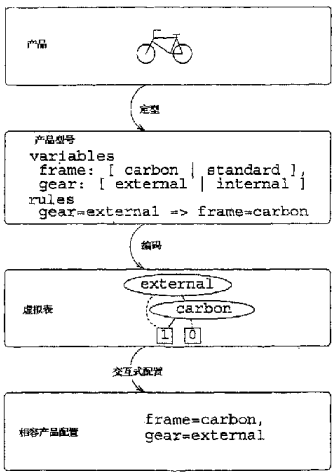
[74] 专利代理机构 中原信达知识产权代理有限责任公司
代理人 顾红霞 钟 强

权利要求书 11 页 说明书 63 页 附图 8 页

[54] 发明名称 利用有向无环图配置产品的方法

[57] 摘要

一个复杂的产品包括多个部件，其中每个部件都和其他部件相互关联。这种相互关联性的一个后果是，一个部件的选择可能和产品中包括的其他部件相排斥。一个相容的配置是一个满足所有内部关联性的部件选择。用于计算机辅助配置的一个计算机程序可以帮助终端用户作出导向相兼容产品的选择。本发明的优选实施例，虚拟制表，是一个用于跟踪大量部件的内部关联性，可以创建一个有效，准确的配置程序的方法。这种程序允许利用网络(如互联网)实现交互式配置。本发明的另一个方面，智能搜索，可以从一个产品数据库里计算出部件之间的一组内部关联性。



ISSN1008-4274

1. 一种配置包含若干部件的产品的方法，该方法包括：

• 为每个部件提供涉及该部件的一组替换件的信息，

• 定义涉及不同部件的替换件之间的兼容性的规则，

• 在一个有向无环图（DAG）中表示规则，及

• 通过重复下列步骤反复地配置产品：

－选定一个部件，

－从该部件的所述替换件组中选出一个替换件，

－校验所述 DGA 以判断所选的替换件是否和其他部件的选定替换件相兼容。

2. 一种如权利要求 1 所述的产品配置方法，其特征在于当每个部件的替换件被选定且部件的选定替换件相兼容时，停止反复配置。

3. 一种如权利要求 1－2 之一所述的产品配置方法，其特征在于选择替换件的步骤，并且在选择替换件之前，包括：

• 利用 DAG 至少为一个部件确定该部件的一个替换件子集，使得该子集中的每个替换件都和其他部件的选定替换件相兼容，及

• 将该信息提供给一个用户。

4. 一种如权利要求 3 所述的产品配置方法，其特征在于该方法还包括向系统提供一个语音合成器，而向用户提供信息还包括

• 提供语音合成器生成的语音信息

5. 一种如权利要求 1－4 之一所述的产品配置方法，其特征在于选定一个部件和替换件的步骤还包括，对每个部件：

• 利用 DAG 校验部件的哪些替换件与其他每个部件的选定替换件中的至少一个相兼容

• 将该信息提供给用户

• 允许用户从与其他每个部件的选定替换件中的至少一个相兼容的替换件中选择一个。

6. 一种如权利要求 1－5 的方法，其特征在于选择替换件和校验 DAG 的步骤还包括以下步骤：

• 选择或定义选定部件的一个替换件子集，

• 校验 DAG 以确定该子集中的哪个替换件与其他部件的选定替换件兼容，及

• 提供涉及子集中那个替换件和其他部件的选定替换件兼容的信息。

7. 一种如权利要求 1－6 之一所述的产品配置方法，其特征在于反复配置还包括：

• 至少一次，定义涉及限制至少一个部件的替换件的信息，以及

• 校验 DAG 以确定部件的哪个替换件与限制信息相兼容。

8. 一种如权利要求 1－7 所述的产品配置方法，其中反复配置是在一个用户的请求下停止的，并提供涉及所有可能兼容产品的信息，所述所有可能兼容产品对每个已选定一个替换件的产品都包括至少一个选定替换件，同时将该信息提供给该用户。

9. 一种如权利要求 1－8 所述的产品配置方法，其中反复配置包括这样的步骤：获得若干所有可能兼容产品，所述所有可能兼容产品对每个已选定一个替换件的产品都包括至少一个选定替换件，并提供该信息给用户。

10. 一种如前述权利要求之一所述的方法，其特征在于在 DAG 中表示规则的步骤包括在一个包括下列成分的图中表示规则：

• 至少一个终端节点，

• 多个节点，包括：

— 有多个可能不相交的结果的数学表达式，及

— 若干与该表达式可能结果数目相对应的指针，其中

• 至少一个节点的指针指向另一个节点，

• 至少一个节点的指针指向所述至少一个终端节点中的一个节点，及

• 至少一个节点是最顶端节点，从该最顶端节点定义了一条或多条从一个最顶端节点到所述至少一个终端节点中的一个终端节点的路径，所述路径通过一个或多个节点及其指针，每个节点都至少是一条路径的一部分。

11. 一种如权利要求 10 所述的产品配置方法，其特征在于在该 DAG 中表示规则的步骤包括向一个或多个节点提供数学表达式，其中每个数学表达式包含一个数学操作符，每个操作符描述被相关节点的指针所指的节点表示的规则如何组合的，以表示组合规则集。

12. 一种如权利要求 10－11 所述的产品配置方法，其特征在于在 DAG 中表示规则的步骤包括在一个包括多个节点的 DAG 中表示规则，其中数学表达式是一个布尔表达式。

13. 一种如权利要求 10－12 所述的产品配置方法，其特征是在 DAG 中表示规则的步骤包括在一个包括多个节点的 DAG 中表示规则，其中每个节点包括一个是变量的数学表达式。

14. 一种如权利要求 10－13 所述的产品配置方法，其特征在于在 DAG 中表示规则的步骤包括在一个包括节点的 DAG 中表示规则，这些节点的数学表达式根据给定次序排序，使得对于每个节点实际节点的表达式次序低于被该实际节点的指针所指的任何节点的表达式。

15. 一种如权利要求 10－14 之一所述的配置产品方法，其特征在于在 DAG 中表示规则还包括以下步骤：

·识别具有相同表达式，且指针指向相同节点的第一和第二节点，及

·将指向第一节点的指针指向第二节点。

16. 一种如权利要求 10－15 之一所述的产品配置方法，其特征在于在 DAG 中表示规则的步骤包括：

·将每个规则表示为一个逻辑表达式，

·根据每个逻辑公式创建一个部分 DAG 来表示该公式的可能解集，

·根据表示每个逻辑公式的部分 DAGs 来创建表示所有规则的 DAG。

17. 一种如权利要求 16 所述的配置产品方法，其特征在于为每个部件提供涉及替换件的信息的步骤包括：

·选择用于表示该部件的各个替换件的布尔变量，

·为该部件的每个替换件提供一个编码，作为布尔变量的布尔值的组合。

18. 一个如权利要求 17 所述的方法，其特征在于将每个规则表示为一个逻辑公式/表达式的步骤包括：提供涉及与规则相关的替换件的布尔变量，并且根据规则使这些变量相互关联。

19. 一种如权利要求 10 所述的方法，其特征在于在 DAG 中表示规则的步骤包括提供至少一种类型的终端节点，并且对于每条包括该终端节点的路径，所有表达式与涉及该路径的指针的所有相关结果的组合涉及一个兼容产品或一个不兼容产品。

20. 一种如权利要求 19 所述的产品配置方法，其特征在于在 DAG 中表示规则的步骤包括提供一个第一类型和一个第二类型的终端节点，其中：

• 对于每条包括一个第一类型的终端节点的路径，所有表达式的与所有涉及该路径指针的相关结果的组合涉及一个兼容产品，及

• 对于每条包括一个第二类型的终端节点的路径，所有表达式与所有涉及该路径指针的相关结果的组合涉及一个不兼容产品。

21. 一个如权利要求 20 所述的方法，其特征在于终端节点的第一类型用来表示“真”，“一”或“1”，及终端节点的第二类型用来表示“假”，“零”或“0”。

22. 一种如权利要求 17 和权利要求 20－21 中任一项所述的方法，其特征在于选择一个替换件的步骤包括：识别涉及该部件其他替换件的布尔变量和包含涉及上述其他替换件的表达式的节点，及在该 DAG 中，识别包含上述节点的路径并将其中任何一个第一类型终端节点改变为第二类型。

23. 一种如权利要求 9 和权利要求 20 所述的方法，其特征在于通过下列施加到 DAG 的步骤来执行对不同选择的可能情况的数目的计算，对于每个最顶端节点：

• 从最顶端节点开始，通过执行如下步骤来反复地查找实际节点所表示的可能情况的数目：

－ 如果该节点是一个终端节点，当终端节点是第一类型时，提供“1”；当终端节点是第二类型时，提供“0”，

－ 否则：查找由该实际节点指针指向的每个节点表示的可能情况的数目，并从中计算该节点表示的可能情况的数目。

24. 一种如前权利要求之一所述的产品配置方法，其特征在于如果选择的替换件与其他选定替换件不兼容，校验 DAG 的步骤还包括：

• 提供涉及与该选择的替换件不兼容的其它选定替换件的信息，及

• 提供该信息给一个用户。

25. 一种如前权利要求之一所述的产品配置方法，其特征在于定义规则的步骤包括：

• 通过查询一个数据库，获得涉及与一个或多个部件的替换件相关的信息，和/或涉及不同部件的两个或多个替换件之间兼容性的信息，及

• 根据从该数据库中获得的信息创建一个或多个规则。

26. 一种如权利要求 25 所述的产品配置方法，其特征在于该数据库包括一个二维表，该二维表的多个行中每一行都包含涉及包括每个部件的替换件的产品的信息，这些替换件都是兼容的，其中提供一个规则的步骤包括提供一个涉及每一行信息的规则，其中在 DAG 中表示规则的步骤包括提供规则的逻辑和。

27. 一种如权利要求 10 所述的产品配置方法，其特征在于校验 DAG 以确定一个替换件是否兼容的步骤包括在该 DAG 中搜索一条从一个最顶端节点到一个终端节点的路径，该搜索包括：

• 从作为一个实际节点的最顶端节点开始，

• 重复下列步骤，直到实际节点实一个终端节点：

— 计算实际节点中的数学表达式，并根据其他部件选定的替换件决定输出结果，

— 选择表示该结果的节点指针，

— 选择被该选定指针指向的节点作为实际节点。

• 提供涉及选定替换件的信息，及

• 涉及该路径的信息表示这些选择是兼容的。

28. 一种如权利要求 20 所述的产品配置方法，其特征在于通过下面的方法从 DAG 中的一条路径提供信息，即，从该路径的节点的表达式来提供关于一个指定部件的哪个替换件已被选定的信息，并且

包括这些替换件的产品的兼容性信息由该路径的终端节点的表示来提供。

29. 一种如权利要求 28 所述的产品配置方法，其特征在于该 DAG 的节点表达式是布尔变量，该终端节点表示或“真”或“假”，路径的信息涉及该路径节点数学表达式中的变量的身份及其中的值或相关性，所述身份和值/相关性涉及部件的选定替换件，其中当该路径的终端节点表示“真”时，选定的部件兼容，当该路径的终端节点表示“假”时，该选定部件不兼容。

30. 一种如权利要求 10 所述的产品配置方法，其特征在于在 DAG 中表示规则的步骤包括：

• 在一个实际 DAG 中表示该规则，

• 选择至少一个要隐藏的部件，

• 通过如下步骤改变所述实际 DAG：

— 在包括与所选择的部件的表达式有关的实际 DAG 中识别节点，

— 从该实际 DAG 中去掉这些节点，

— 增加不包括涉及该选择的部件表达式的节点至实际 DAG，使得这些部件所隐含的兼容性被该实际 DAG 所反映，

• 提供该实际 DAG 作为表示该规则的 DAG。

31. 一种如权利要求 10 所述的产品配置方法，其特征在于在 DAG 中表示规则的步骤包括：

• 对于每个规则，创建一个表示该规则的部分 DAG，

• 识别至少一个要隐藏的部件，

• 选择被识别部件的一个次序，

• 初始化创建一个未表示规则的实际 DAG，然后重复下列步骤：

— 选择一个次序最低的未选择部件，

— 重复如下步骤，直到包括涉及所选择部件的表达式的所有的

部分 DAGs 都被选定：

* 选定一个包括涉及所选择的部件表达式的部分 DAG，

* 将该实际 DAG 和选定部分 DAG 组合成一个信的实际 DAG，

— 通过如下步骤改变该实际 DAG：

* 在包括涉及已识别部件的表达式的实际 DAG 中识别节点，

* 从该实际 DAG 中去掉这些节点，

* 增加不包括涉及识别部件表达式的节点至该实际 DAG，以便识别部件所隐含的兼容性可以通过该实际 DAG 反映，

•通过将该实际 DAG 和所有未选定部分 DAGs 合并来提供 DAG。

32. 一种如前权利要求之一所述的产品配置方法，该方法还包括：

• 识别一个用户

• 用户通过由其控制的设备和执行反复配置的设备之间的通讯来执行选择一个部件替换件的步骤，

• 发送有关校验该 DAG 的信息给该用户。

33. 一种如前权利要求之一所述的产品配置方法，其特征在于该方法还包括：

• 识别一个用户，

• 在反复配置前：

— 发送该 DAG 到由用户控制的设备里，

— 在用户设备里执行反复配置。

34. 一种如前权利要求之一所述的产品配置方法，在反复配置中，还包括如下步骤：

• 获取有关没有选定替换件的部件的一个或多个替换件信息，该一个或多个替换件中的每个替换件都和已选定的替换件相兼容及

• 提供该信息给用户。

35. 一种如前权利要求之一所述的产品配置方法，其特征在于该方法还包括向系统提供一个语音识别器，其中反复配置产品的步骤还包括：

• 从一个语音合成器识别的文本中选择一个部件，及

• 在该语音识别器识别的文本中，从该部件的替换件组中选择一个替换件。

36. 一种如前权利要求之一所述的产品配置方法，其特征在于该方法还包括识别一个可配置设备和一个接口设备，及

• 将表示规则的 DAG 存储在所述可配置设备里，

• 将 DAG 从该配置设备上传到该接口设备，及

• 在反复配置产品的步骤里，在该接口设备上执行校验 DAG 以判断选择的替换件是否和其它部件的其它选定替换件兼容。

37. 一种如权利要求 36 所述的产品配置方法，其特征在于该方法还包括识别所述可配置设备中的一列预定部件，以及识别所述可配置设备中这些部件的一列预定替换件，其中反复配置产品的步骤还包括：

• 在接口设备上执行对 DAG 的校验以判断选择替换件是否与其他部件的其他选定替换件兼容以及是否与预定替换件兼容。

38. 一种如前权利要求之一所述的产品配置方法，其特征在于该方法还包括识别一列观测者部件和一列非观测者部件，及

• 在一个 DAG 中，表示非观测者部件的规则，

• 为每个观测者部件确定所述规则的一个子集，以便从这些规则里，可能为与非观测者部件的替换件兼容的观测者部件确定替换件，

• 为每个观测者部件，将所述规则子集表示为一个观测者 DAG，及

• 在反复配置产品的步骤里

— 校验 DAG 以判断选择的替换件是否和其他部件的其它选定替

换件兼容，

— 通过为每个部件确定是否只有一个替换件与所有选定替换件兼容，来来确定一个系统确定的替换件的集

— 对至少一个观测者部件，为该观测者部件校验观测者 DAG 以确定是否只有一个替换件同其他选定替换件及所述系统确定的替换件集相兼容，及

— 提供该信息给用户。

39. 一种如前权利要求之一所述的产品配置方法，其特征在于反复配置产品的步骤还包括：

• 对于每对部件与替换件，提供该对状态的分类，

• 将该分类归入到包括封锁类，可选择类，用户选择类，系统选择类与强制类的一个结果列表中的一种，

• 当即使是不考虑其他部件替换件的选择的情况下，也不能为该部件选定一个替换件时，该状态规定为封锁类型，

• 当该部件的替换件与其他部件的选定替换件都相兼容时，该状态规定为可选择类型，

• 当为该部件已经选定替换件时，该状态规定为用户选择类型，

• 当替换件是该部件里与其他部件选定替换件相兼容的唯一替换件，且用户还没有选定该替换件时，该状态规定为系统选择类型，

• 当该部件的可选替换件与其他部件替换件的一些选择不相兼容时，该状态规定为强制类型，

• 将分类信息提供给一个用户。

40. 一种包含计算机程序代码装置的计算机程序，当该程序在一台计算机上运行时，适用于执行前述任何权利要求的方法的所有步骤。

41. 如权利要求 40 所述的，包含在计算机可读介质上的一个计算机程序。

42. 一种包括如权利要求 40 所述的计算机程序的计算机可读介质。

利用有向无环图配置产品的方法

技术领域

本发明涉及配置多个部件组成的产品的方法。这些部件之间具有相互关联性，因此需要对各个部件进行合理的选择以提供一个能运行的产品。一般而言，确定一个共同运行的部件集是一项复杂的任务，通常需要计算机程序的参与辅助。这种计算机程序必须采用有效、准确的方法来处理部件之间的相互关联性。本发明涉及如何实现该计算机程序。

技术背景

本发明涉及一种进行计算机辅助产品配置的方法。

一个复杂的产品由多个部件组成。一般，通过将一个产品看成是由几类部件组成来形成一个复杂产品的产品型号。对于每一类部件来说，都有一组特定的替换件。

这里举出一个自行车产品型号的例子。一个自行车是由以下几个部件组成：一个车架、一个前轮、一个后轮和一组齿轮。对于车架，存在几种替换件：男式碳素型、女式标准型、男式标准型、越野型。对于前轮部件，有光滑型和越野型。对于后轮部件也存在光滑型和越野型。最后，对于齿轮组也有内部三倍速型和外部十倍速型。

在配置的语境中，“部件”不应仅被理解为一个物理部件的一般描述。部件也可以是诸如颜色、形状等的属性，或者齿轮数、马力等参数。一个部件也可以理解为表达用户需要、而非产品特征的“需求属性”，如自行车的类型（越野型、都市型、重载型等）、用户的品味（时髦型、经典型、童真型）、价格、重量或是产品用户的兴趣所在

的类似属性。

为了生产一个复杂的产品，必须为每一部件选择一个特定的替换件。若干选择被称为产品的部分配置。对每一部件都选定替换件的完整选择被称作为产品的一个完全配置（或一个配置）。

随着产品部件数量的增加，产品的可选性配置方案数目迅速增加。例如，在自行车例子里，一个人必须在四种框架，两种前轮，两种后轮以及两种齿轮组中进行选择。因此，存在 4×2×2×2＝32 种不同的配置。在现实生活中，这类的选择配置方案可以快速增加到百万以上。

由于不兼容等原因，不是所有的配置方案都能够正常工作。如果还是以自行车为例，可能前轮和后轮必须是同种型号。另一个需求是，只有男式炭素型车架才能允许配置外部十倍速齿轮组。这些替换件中不兼容性的描述，被称作是产品要求。产品要求通常由定义部件之间兼容性的规则表示。当满足所有要求时，这个配置是相容的。对自行车的例子，在 32 种可能的配置中有 10 种不同的相容性配置（8 种带内部三倍速齿轮组的配置和 2 种带外部 10 倍速齿轮组的配置）。

一般来说，这些要求都很复杂，难以由人来全面检验。确定一个相容性配置是一项复杂的任务。一个计算机程序在配置过程中可以起很大的作用。通常，计算机程序通过用规则检验用户的选择来工作。这种检验一般很难执行：或者检验耗费的时间过长，或者检验结果的的不准确。至少有两种方法可以用来处理这些规则：

显式列举法

这种方法通常用位向量来表示所有可能的相容性配置。所有的可能配置都通过规则来校验，校验出来的相容性配置列举在一个列表里，通常使用位向量哈希表。这种方法的一个关键缺陷是，配置的数

目随着可用部件数量增加而迅速增加（典型的是，配置数目随着部件数量的增加而指数级增加）。这意味着需要极大的存储容量，且该方法也不适用于大型产品型号。这种方法的另一个问题是，即使配置数量足够小到可以保存在内存中，用来遍历和处理每种可能配置的算法的运行时间也随配置数目而线性增长。

规则/约束传播法

在作出一个配置选择时，需要检索规则数据库以校验该配置的相容性。检索的时间难以预期，而为了保证对用户的及时响应，通常对消耗的时间有限制。为了满足时间限制，检索经常会在没有找到完全、准确结果的情况下，提前结束。通过重复运用规则库中的规则到作出的选择上，该检索建立在信息积累的基础上。这通常都很费时，费力。而且，检索时间以及检索的质量，都很大程度上依赖于如何形成规则。

现有技术的工具应用上述两种方法。它们被开发为销售助理工具，现已应用于互联网。在互联网上，没有一个人类销售助理可解决信息的误差和缺失。用户必须自己来完成整个销售过程，因而也加重了对销售系统质量的需求。销售系统必须对用户的需求有快速反应，并保证结果的准确性。例如，不允许出现和用户之前选择不一致的替换件（即，破坏了一些规则）。

现有技术的工具很难在保证需要的响应时间的同时获得准确的结果（在系统处理多个并发用户及复杂产品的同时）。

一些专利涉及产品配置：

US6,115,547 披露了一种通过缓存早先配置的特定方法来增加配置程序性能的产品配置法。

US5,675,784 披露了一种包括用于模型化产品的数据结构（三层分级数据结构）的产品配置法。

EP 0770239B1 披露了一种包括专家系统的产品配置法。该配置法涉及规则/约束传播法。

US 5,206,949 披露了一种包括数据库搜索和检索系统的产品配置法。

US5,844,554 披露了一种包括用于设计产品型号的用户图形界面的产品配置法。

US5,987,473 披露了一种包括通过网络进行的交互式配置的产品配置法。

US5,995,979 披露了一种包括允许用户通过网络在数据库中选择条目的方法的产品配置法。

US5,745,765 披露了一种包括允许用户选择相容性配置的方法的产品配置法。

本申请描述的发明应用在一种在硬件电路形式验证中已知的符号模型校验（symbolic model checking）来解决在开发计算机辅助配置中的计算问题。符号模型校验参见[K.L McMillan Symbolic Model Checking: An Approach to the State Explosion Problem]。

发明内容

本发明的第一方面，涉及一种配置包含多个部件的产品的方法，该方法包括：

- 为每个部件提供有关该部件的一组替换件的信息；
- 定义关于不同部件的替换件之间兼容性的规则；
- 在有向无环图 (DAG) 中表示上述规则，及

- 通过重复以下步骤反复地配置产品

—选择一个部件

—从该部件的替换件组中选择一个替换件

—校验所述 DAG 以判断选定的替换件是否和其它部件的选定替换件兼容。

在这里的语境中，部件不应仅被理解为一个物理部件的一般描述。部件也可以是诸如颜色、形状等的属性，或者齿轮数、马力等参数。一个部件也可以理解为表达用户需要、而非产品特征的“需求属性”，如自行车的类型（越野型、都市型、重载型等）、用户的品味（时髦型、经典型、童真型）、价格、重量或是产品用户的兴趣所在的类似属性。

一个规则可能涉及到来自，如一个产品中两个不同部件中的一个替换件的兼容性。但是，规则最好涉及到更多数量部件中替换件的兼容性。在一个极端的情况下，当然也不是不可能发生的，一个规则涉及到包括每一部件的替换件的产品。

自然，涉及到一个或是一组替换件的信息可能是有关相似性和区别性的信息。通常，这些信息与另一部件和/或另一部件的替换件相关。

当在 DAG 中表示了规则时，不再需要对规则（通常是大量的规则）进行校验。相反，根据与选择/选定替换件有关的信息来遍历、分析，甚至是修正 DAG。这种方法比逐个校验若干规则快很多。

在这里的语境中，一个替换件被“选定（chosen）”是指它被“选择（selected）”为需要进行配置的组合起来的产品的一部分。通常，是当这个替换件至少和一个选定的替换件相兼容时。

当每个部件都选定好替换件，或者优选的是，当这些部件的选定

替换件之间互相兼容时，可以结束这种反复配置。

在选择一个替换件前，需要利用 DAG 为至少一个部件来确定一组替换件子集，以便该子集中的每一替换件可以和其他部件的选定替换件相兼容，并将这些信息提供给用户。

在这种情况下，用户需要关于一个给定部件与一组替换件的兼容性信息——和已选定的替换件相兼容性的信息——以便从该组里确定一个兼容的选择。该子集可以和用户的偏好有关，如尺寸、颜色、制造商、生产地等。

如果计算机用语音把这些信息提供给用户时，与系统进行交互将更容易。这是通过向系统提供一个语音合成器并向用户提供信息来实现的，给用户提供信息还包括：

• 提供通过语音合成器产生的语音信息。

或者，选择一个部件以及为每个部件选择一个替换件的步骤还包括：

• 利用 DAG 来校验一个部件的哪一个替换件与其他每一部件（即已经选定替换件的那些部件）的选定替换件中的至少一种相兼容，

• 提供该信息给用户，

• 允许用户在至少与每个其他部件的选定替换件之一相兼容的替换件中选择一个替换件。

因此，在该方法里，提供涉及该部件的所有替换件与已选定的替换件的兼容性信息，使得用户很快就能进行产品配置。

但是，相反地或附加地，选择一个替换件与校验 DAG 的步骤最好还包括：

• 选择或定义选定部件的一个替换件子集，

• 校验 DAG 以确定该子集中何种替换件与其他部件的选定替换件相兼容，及

• 提供有关该子集中何种替换件与其他部件的选定替换件相兼容的信息。

这尤其适用于用户还没有确定一个特定替换件，但他提供一个已被校验过与其他部件的选定替换件兼容性的替换件子集的情况。这些信息可以用于在配置中进一步引导用户。

另一种好的办法是：

• 至少定义一次有关限制至少一个部件的替换件的信息，及

• 校验 DAG 以确定这些部件的何种替换件与限制信息相兼容。

这种限制信息可以由用户提供，关于部件的替换件中何种替换件与限制信息相兼容的信息也可以提供给用户。

这种限制信息可以是有关用户所需的不同组的替换件之间兼容性的信息。

这种反复的配置也可以基于用户的指令而终止，通常是在还没为每个部件选定/选择一个替换件，或是选择/选定的替换件不能完全兼容的时候。然后，有关包括为每个产品都至少选定一个替换件的所有可能的兼容性产品的信息被提供给用户。

因而，用户可以终止配置，得到包括已选定替换件的可用相容产品总数目的信息。

反复配置可以包括如下步骤，即获得所有可能兼容的产品的数目，其中可能兼容的产品对每个已为其选定一个替换件的产品都包括

至少一个选定替换件，并提供该信息给用户。这种方式，用户可以被不断被告知包括已选定替换件的产品的数目。应注意，用户能够为一个指定的部件实际上选择或选定多个替换件。在这种情况下，相容性的校验将是对每一替换件的相容性校验，潜在最终产品的总数与包含这些替换件之一的潜在最终产品数目有关。

通常，在一个 DAG 中表示规则的步骤可以包括在一个包括下列项的图中的表示规则：

- 至少一个终端节点
- 多个节点，包括：
  - 一个存在多个可能不相交结果的数学表达式及
  - 多个与该表达式可能结果数目相对应的指针，

其中：

- 至少一个节点的指针指向另一个节点，
- 至少一个节点的一个指针指向至少一个终端节点中的一个节点，及
- 至少一个节点是最顶端节点，从该最顶端节点定义了一条或多条经一个或多个节点和指针、从一个最顶端节点到所述至少一个终端节点中的一个的路径，每个节点都至少是一条路径的一部分。

这是一个在 DAG 中表示规则的标准方法。因此，这些规则被表示成数学公式，并引入一个或多个节点。每个规则都包含一个或多个结果——并且每个节点的指针都与一个上述结果相关。因此，规则的不同结果能提供通过图/DAG 的不同路径的遍历。

因此，在 DAG 中表示规则的步骤可以包括向一个或多个节点提供数学表达式，每个数学表达式包含一个数学操作符，每个操作符描述被相关节点指针指向的节点表示的规则是如何组合的，以便代表组合的规则集。

在 DAG 中表示规则的步骤可以包含在一个包括多个节点的图中表示规则，该图的数学表达式是一个布尔表达式和/或一个变量。

而且，在 DAG 中表示规则的步骤中，也可以包括在一个包括节点的图中表示规则，该图的数学表达式根据指定次序排序，使得对每个节点，一个实际节点的表达式在次序上低于该节点的指针所指的任何节点的表达式。

提供一个顺序有助于 DAG 中的若干操作，如在一个 DAG 中搜索和合并两个 DAG。

为了维护一个适合的 DAG，在 DAG 中表示规则还包括以下步骤：

- 识别具有相同表达式、且指针都指向相同节点的第一和第二节点，
- 使指向第一节点的指针指向第二节点。

在那种情况下，两个代表同样内容的节点可以简化成一个节点。

提供 DAG 的一个优选方法，其中在 DAG 中表示规则的步骤包括：

- 用逻辑表达式表示每个规则，
- 从每个逻辑公式构建一个代表该公式的可能解集的部分 DAG，
- 由表示每个逻辑公式的部分 DAG 来构建表示所有规则的 DAG。

该方法相当简单，在于从一个规则里构建一个部分 DAG 通常是一个容易的工作，且合并 DAG 也是一项公知的技术，实际上如果利用上述的表达式次序是便利的。

优选地，提供有关各个部件替换件信息的步骤包括：

• 选择用于表示该部件的各个替换件的布尔变量，

• 为该部件的每个替换件提供一个编码，作为布尔变量的布尔值组合。

将每个规则表示为一个逻辑公式/表达式的步骤包括：提供涉及规则相关的替换件的布尔变量，并根据规则使这些变量关联。

一般，在 DAG 中表示规则的步骤优选包括提供至少一种类型的终端节点，并且对包含这种终端节点的每条路径，所有表达式与所有涉及该路径的的指针的相关结果的组合涉及一个相兼容产品或不相兼容产品。

由以上内容看出，一条路径上节点的数学表达式的变量和部件的多个替换件有关。很清楚，路径由连接节点的指针所定义，其中每个指针都关联这一个数学表达式的结果，以及给定的变量之间的关系。因此，路径的信息——包括终端节点的信息——优选提供关于产品、替换件以及替换件之间兼容性的信息。

优选地，在 DAG 中表示规则的步骤包括提供一个第一类型和一个第二类型的终端节点，其中：

• 对每条包括第一类型终端节点的路径，所有表达式与所有涉及该路径指针的相关结果的组合涉及一个相兼容产品，并且

• 对每条包括第二类型终端节点的路径，所有表达式与所有涉及该路径指针的相关结果的组合涉及一个不兼容产品。

此时，终端节点的第一类型可用于表示“真”，“一”或者“1”，终端节点的第二类型可以用来表示“假”，“零”或“0”。

一般来说，选择一个替换件的步骤可包括识别关于该部件其他任何替换件的布尔变量和包括涉及其他替换件的表达式的节点，以及在DAG中识别包含上述节点的路径并将其中任何一个第一类型终端节点改变为第二类型。因此，所述路径可直接关联一个“不兼容产品”，由于这些产品已不令人感兴趣——选定的替换件通常与同部件的其他替换件不相兼容。如果用户为该部件选择了一个替换件子集，那么对于不在该子集中的其他部件替换件，自然也是接着同样的过程。

此时，计算不同选择的可能情况的数目可以通过如下应用于DAG的步骤执行.，对每个最顶端节点：

• 从最顶端节点出发，通过执行以下步骤反复地查找实际节点表示的可能情况的数目：

— 如果该节点是终端节点，若它是第一类型，就提供“1”，若它是第二类型，就提供“0”，

— 否则：查找由该实际节点的指针指向的每个节点所表示的可能情况的数目，并从中计算出该节点所表示的可能情况的数目。

通常，一个具有例如由一个指针表示的可能情况数目和另一个指针表示的第二数目的可能情况数目的节点来表示的可能情况数目，可以计算为第一数目的可能情况和第二数目的可能情况之和。但是，如果因为DAG的尺寸减小（诸如“本地减小 local reduction”），隐节点（隐藏地）位于实际节点和第一和/或第二节点所指的节点之间，这些隐节点在查找实际节点表示的可能情况数目时也要考虑在内。

如果，在配置中，一个选择的替换件和其他选定替换件不相兼容，校验DAG的步骤还可以包括：

• 提供涉及与该选择的替换件不相兼容的其他选定替换件的信息，及

• 将该信息提供给一个用户。

此时，用户可能会实际输入或选定/选择该选择的替换件，然后撤销与该替换件不相兼容的其他替换件。

现有很多方法可以提供涉及兼容性的规则。一个优选的方法，其中至少有一个规则通过如下方式来定义：

- 通过查询一个数据库，获得涉及一个或多个部件的替换件的信息，和/或涉及不同部件的两个或多个替换件之间的兼容性信息，及
- 利用从数据库中获得的信息创建一个或多个规则。

一个简单的实现方法是，该数据库包括一个两维表，在该两维表每一行里包括关于一个每个部件都有一个替换件的产品的信息，这些替换件都相互兼容，其中提供一个规则的步骤包括提供一个涉及每一行信息的规则，在 DAG 中表示规则的步骤包括提供规则的逻辑和（disjunction）。

因此，该表的每一行包含涉及为每个部件都选定替换件的一个完整产品的信息，其中每个产品的所有替换件都相容。一个单独的行里的信息可以提供为一个随后要导入 DAG 的规则。

从提供产品配置的实体的角度来看，这种方法的优点在于，由于规则仅仅和已识别产品预先确定的范围有关，只对这些产品进行配置。因此，尽管从进行配置的用户角度来看，配置除了兼容性外，没有任何限制，但是这些配置最后都会产生一个供应商已经识别的产品。

优选地，校验 DAG 以判断一个选择的替换件是否和选定替换件兼容的步骤包括，在该 DAG 中搜索一条从最顶端节点到终端节点的路径，该查找包括：

- 从作为一个实际节点的最顶端节点开始，
- 重复以下步骤，直到实际节点是一个终端节点：

— 估算实际节点中的数学表达式并根据其他部件的选定替换件确定该结果，

— 选择表示该结果的节点的指针，

— 选择一个被选定指针所指的节点作为实际节点。

- 提供和选定替换件相关的信息
- 有关路径的信息表示该选择是相兼容的。

一个从该 DAG 中的一条路径里提供信息的简单方法是，从该路径的节点的表达式中，提供有关一个给定部件的哪个替换件已被选定的信息，并且包括这些替换件的产品的兼容性信息由该路径的终端节点的表示来给定。

因此，有关单个替换件的信息来自节点的表达式以及连接节点的指针，通过路径的终端节点可以得到兼容性信息。

优选的方式是，涉及 DAG 中节点的表达式是布尔变量，该终端节点表示“真”或“假”，一条路径包含一个或多个有一个数学表达式和一个指向路径中另一个节点或终端节点的指针的节点，涉及该路径中节点数学表达式中变量的身份（identities）的路径信息，及其中的值或相关性，所述特性、值/相关性涉及部件的选定替换件，如果该路径的终端节点表示“真”，则选定部件是兼容的，如果该路径终端节点表示“假”则该选定部件是非兼容的。

在配置中，要考虑一种一个部件是一个类型的特殊情况，但这种情况可能不提供信息，或者不与如执行配置的用户相关。因此，需要在配置中“隐藏”这种部件。

一个可以隐藏的部件的例子是自行车轮中轮毂的宽度。这种宽度很重要，因为它描述了车架和车轮的兼容性，但一个配置自行车的用户不需要注意这点。系统可以简单地隐藏这个部件并确保用户不能执

行与隐含地选定的轮毂宽度不兼容的配置（如已选定车架或车轮所定义的）。

在那种情况下，在 DAG 中表示规则的步骤可以包括：

- 在一个实际 DAG 中表示规则，
- 选择至少一个要隐藏的部件，
- 通过以下方式改变所述实际 DAG：

— 在包括与选择的部件的表达式有关的实际 DAG 中识别节点，

— 从该实际 DAG 中去掉这些节点，

— 增加不包括涉及选择的部件的表达式的节点至实际 DAG 中，以便这些部件隐含的兼容性可被实际 DAG 反映出来，

- 将该实际 DAG 作为表示规则的实际 DAG。

因此，在一个方法中简单地改变该 DAG，使得一个为其他部件隐含地选择替换件的隐藏部件的一个替换件隐含地为其他部件选择替换件，所以，尽管用户不能校验，但随后的兼容性校验也会涉及到“隐藏”部件。

在修改 DAG 中，优选方式是尽快去掉这些“隐藏部件”。这可通过如下方式实现：

- 对每一规则，创建一个表示该规则的部分 DAG，
- 识别至少一个需隐藏的部件，
- 选定被识别部件的一个次序
- 初始化创建一个未表示规则的实际 DAG，然后重复：

— 选择一个次序最低的未选择部件，

— 重复如下步骤，直到包含涉及选择的部件表达式的所有部分 DAG 都被选定：

* 选定一个包含涉及所选择的部件表达式的部分 DAG，

* 将实际 DAG 和选定的部分 DAG 合并成一个新的实

际 DAG，

— 通过下列步骤改变实际 DAG：

* 在包含涉及已识别部件的表达式的实际 DAG 中识别节点，

* 从实 DAG 中去掉这些节点，

* 增加不包含涉及已识别部件的表达式的节点至实际 DAG 中，使得该实际 DAG 反映已识别部件之间隐含的兼容性，

• 通过将实 DAG 和所有未选定的部分 DAG 合并成一个 DAG 来提供 DAG。

通常，这种方法还可包括：

• 识别一个用户，

• 用户通过由其控制的设备和执行重复配置的另一设备之间的通讯，来执行选择一个部件替换件的步骤，

• 发送有关 DAG 校验的信息给该用户。

因此，计算的主要负荷（派生于规则与 DAG，及 DAG 的反复校验）是在远离用户的地方执行，并仅仅是将结果发送给用户。这就节约了用于执行配置各种类型产品的带宽，如互联网的带宽。

这种方法还可包括：

• 识别一个用户，

• 在反复配置前：

— 发送 DAG 到该用户操作的一个设备里，

— 在该用户的设备里执行反复配置。

在这种方法里，DAG 被发送给用户，然后在客户端的 DAG 上执行配置——这可以是在一个该用户控制或拥有的计算机上执行。

一个特别优选的实施例是，在反复配置过程中，包含以下步骤：

• 获取有关没有选定替换件的部件的一个或多个替换件信息，该一个或多个替换件中的每个替换件都和已选定的替换件相兼容，及

• 提供这一信息给用户。

因此，由于只有合理的替换件才被显示，则大大的加快了该配置的执行且没有用户由于试图结合不兼容的替换件所带来的错误。

使用户和产品配置进行交互的一个较好方法是，提供一个语音合成器给系统，其中，反复配置产品的步骤还包括：

• 从语音合成器识别的文本里选择一个部件，及

• 在语音合成器识别的文本里，从该部件的替换件组中选择一个替换件。

在这种方法里，通过语音来选择替换件，通过电话来进行诸如产品配置的应用是优选的。

在被配置的产品是一个设备的应用中，优选的方式是，这个方法还包括识别一个可配置设备和一个接口设备，及：

• 将表示规则的 DAG 存储在该可配置设备上，

• 从可配置设备里上载 DAG 至接口设备，及

• 在反复配置产品的步骤中，在接口设备上执行校验该 DAG 以判断选择的替换件是否与其他部件的其他选定替换件相兼容。

在这种方法里，涉及配置设备的所有信息都可以存储在该设备里，并且从其他的任何接口设备里进行存取，而这些接口设备不需要特别具备配置设备的知识。

当可配置设备自己就能确定一些替换件时，该方法还包括下面步骤是有益处的，即，识别可配置设备里的一列预定部件并识别可配置

设备里的这些部件的一列预定替换件，其中反复配置产品的步骤还包括：

• 在接口设备上执行对 DAG 的校验以判断选择的替换件是否与其他部件的其他选定替换件相互兼容以及是否与预定的替换件相互兼容。

预定替换件使得用户更容易进行配置，因为需要作的替换件选择少了。

在许多产品的产品配置里，一些部件是观测者部件（observer component），即用户并不会选择这个部件的替换件，而仅仅是对它的兼容性的值有兴趣，这样是有益处的。这能够加以利用，如果该方法还包括识别一列观测者部件和一列非观测者部件，及：

• 在 DAG 中表示观测者部件的规则，

• 为每个观测者部件确定所述规则的一个子集，从这些规则里，可能为观测者部件确定和其他非观测者部件的替换件相兼容的替换件，

• 为每个观测者部件，将所述规则子集表示成一个观测者 DAG，及

• 在反复配置产品的步骤中

— 校验 DAG 以判断选择的替换件是否和其他部件的其他选定替换件兼容，

— 通过为每个部件确定是否只有一个替换件与其他所有选定替换件相兼容，来确定一个系统确定的替换件的集。

— 对至少一个观测者部件，为该观测者部件校验观测者 DAG 以确定是否只有一个替换件同其他选定替换件及所述系统确定的替换件集相兼容，及

— 提供该信息给用户

在不同的 DAG 中表示规则是有利的，因为这种方法减少了 DAG

的整个规模，从而减少存储要求和增加性能。

如果反复配置产品还包括以下步骤，更多有用的信息可以提供给用户：

• 为每一对部件和替换件提供分该对状态的分类，

• 将该分类归于包括封锁类，可选择类，用户选择类，系统选择类与强制类几种结果的列表里的一类，

• 当即使是不考虑其他部件替换件的选择的情况下，也不能为该部件选定一个替换件时，该状态规定为封锁类型，

• 当该部件的替换件与其他部件的选定替换件都相兼容时，该状态规定为可选择类型，

• 当为该部件已经选定替换件时，该状态规定为用户选择类型，

• 当替换件是该部件里与其他部件选定替换件相兼容的唯一替换件，该状态规定为系统选择的类型，

• 当该部件的可选替换件与其他部件替换件的一些选择不相兼容时，该状态规定为强制类型，

• 将分类信息提供给一个用户。

通过向用户提供关于替换件的可能选择的效果的信息，这种分类可以在用户界面里使用。一些替换件是不可能实现的，一些可以直接选择，还有一些已经被用户或系统选择，最后还有一些强制类的，意味着当用户愿意取消一些之前的选择时，可以选择这些替换件。

本发明的第二部分涉及到一个包含计算机程序代码装置的计算机程序，当所述程序运行在一个计算机上时，可以执行上述方法中的所有步骤。

本发明还涉及在计算机可读介质上实现的该计算机程序以及包括该计算机程序的计算机可读介质。

附图说明

下面，联系附录和以下图形对本发明的一个优选实施方式进行详细说明：

图 1 示意了配置过程概览，

图 2 示意利用 ConfigIt Studio 创建产品型号，

图 3 示意一个 PC 的交互式配置，

图 4 示意一个 PC 实例，显示表示第三规则的一个 BDD，

图 5 示意另一个 PC 实例，显示表示域约束的一个 BDD，

图 6 示意另一个 PC 实例，显示表示规则的一个 BDD，

图 7 示意另一个 PC 实例，显示代表包含公有和私有变量的所有规则和域约束的一个 BDD，

图 8 示意另一个 PC 实例，显示仅包括全局变量的、带一个表示规则和域约束的 BDD 的虚拟表，

图 9 示意另一个 PC 实例，显示一个表示基于选择 Seagate-Barracuda-9-9,1GB 硬盘的相容性配置的 BDD，及

图 10 示意另一个 PC 实例，显示一个除 X0 和 X1 外，所有变量都作存在数量词操作（existentially quantified out）的虚拟表。

附件 A 里给出了“产品说明”的一个优选实施例（一个 XML 文档类型声明）

在附件 B 里给出了多个算法的优选实施例：

算法 1：基本 BDD 操作

算法 2：MULTIAPPLY，在一组顶点上应用一个操作

算法 3：MULTIEXIST，一组变量的存在数量词

算法 4：ORDERRULES，根据私有变量对规则排序

算法 5：CONJOINEXISTS，将 BDDs 和作存在数量词操作的变量结合

算法 6：VIRTUALIZETABLE，创建表示一个表的一个 BDD

算法 7：CONFIG1，限制一个选择的一个虚拟表

算法 8：CONFIGCONSISTENT，限制一个选择列表的一个虚拟表

算法 9：CONFIGCHECK，限制一个选择列表的一个虚拟表，确保非空

算法 10：CONFIGIT，限制一列兼容的选择的一个虚拟表，为剩余的产品变量选择相兼容的值

算法 11：CONFIGOUNT，统计一个虚拟表中兼容配置的数目

算法 12：DETERMINEDOMAIN，为一个虚拟表中的被展平（flattened）的变量确定可能值

算法 13：CONFIGCLIENT，交互配置，客户

算法 14：CONFIGSERVER，交互配置，服务器。

具体实施方式

本发明将利用应用在包括多个部件的复杂产品的交互式计算机辅助配置的优选实施例来描述，这也是本发明所解决的问题起源。但是，本领域的技术人员应理解，本发明并不只限于这一特定的应用，而是在实现配置的方法及配置产品的结构方面有更广阔的应用面。

本发明包含一种用于配置产品的方法。由于本发明没有约束，因此一个产品型号可以用来对该产品的相关方面定型。在产品型号中，产品包括多个部件，且每个部件都有一组替换件。每个部件通常都有描述部件相关方面的属性，如颜色，行为，重量，界面等。对每个属性而言，都有一组具体数值。例如，颜色属性里可以有红色，蓝色和绿色。而且，还有涉及到不同部件的替换件之间兼容性的规则。

用于配置产品的方法包括：

• 指定产品的相关方面作为产品型号。产品型号描述部件，部件属性，以及每个部件的替换件和每一属性的值。而且，产品型号还包括一组涉及到部件和属性之间兼容性的规则。

• 将这一产品型号编码成一个表示产品型号兼容配置的虚拟表。

• 利用该虚拟表配置产品产生一个兼容配置。典型的情况是，这些过程是一个在一个用户和一个配置程序之间交互的对话中完成的。

图 1 大致描述了这些步骤。该图示意了一个特殊产品（一辆自行车），一个用于产品型号的特定表（一个文本化描述），一个特定的虚拟表（一个布尔决策图），一个特定交互配置过程。本领域的技术人员应明白本发明并不仅仅约束于这些特定的选择。

• 首先，要产生一个具体产品的产品型号，现以自行车为例。这个具体产品型号捕捉到存在两种不同的框架，还有两种不同的齿轮。而且，该产品型号利用一个规则捕捉到如果选定外部齿轮组，那么框架一定是碳素框架。

• 该产品型号被编码为一个虚拟表。该表是一个表示所有兼容性配置的有向无环图。这个具体的有向无环图是一个带两个变量－external（表示选定外部齿轮组）和 carbon（表示选定碳素框架）的布尔决策图（BDD）（在符号模型检测领域中技术人员所知的）。通俗的说，BDD 和产品型号的联系是：当且仅当 external 和 carbon 的布尔值分配导向终端 1 时，相应的配置是兼容的。

• 现执行该自行车的一个计算机辅助配置。计算机程序显示每个部件可能的替换件。该计算机程序的用户选择一个部件，并为这个部件选定一个可能的替换件。例如，该用户可以选择齿轮组部件，且该齿轮组部件是外部的。基于用户的选择，计算机程序使用虚拟表用于寻找可以导向兼容配置的下一个选择。例如，计算机程序可以利用虚拟表来确定一个外部齿轮组的选择是否意味着框架必须要是碳素的。继续该交互式过程，直到为每个部件选定一个替换件。这一配置过程的结果是一个兼容产品配置。

下面三个部分中将对产品型号、编码过程、和最终配置过程进行深入的描述。在每个部分中，都给出了优选实施例。

产品型号

一般，产品型号用于描述产品是由哪些部件组成以及这些部件之间的相互关联性。

本发明没有在产品型号上设定特定的约束。但是，不作为对本发明的限制，产品型号通常会定义一组产品变量，每个变量所属的域，以及一组规则。每个产品变量表示一个部件或一个属性。对一个表示部件的产品变量，产品变量的域和该部件的可能替换件相应。对一个表示属性的产品变量，产品变量的域和该属性的可能值相应。产品变量可能包括分散域，也包括连续域。部件和属性之间相互关联性表示为规则，并通常制定为产品变量的公式。

一个产品型号的例子是一个计算机产品型号，包括一个主板（三种不同替换件），一个 CPU（两种替换件）和一个硬盘（两种替换件）。由于 CPU 是通过插槽同主板相连，因此插槽的类型是 CPU 和主板的一个重要属性。由于硬盘是通过一个特定类型的控制器和主板相连，所以控制器的类型也是一个重要的属性。下面是一个计算机产品型号的文本化实例：

```
types
    cpu-slot-t = [ SLOT-1 | SLOT-A],
    controller-t = [ IDE | SCSI]

variables
    public motherboard: {
    public name:
```

```
        [ Abit-BX6-ATX | Aopen-AX6BP-ATX | Aopen-AK-72-
KX133-ATX],
          private slot: cpu-slot-t,
          private controller: controller-t
        }

        public harddisk:{
          public name:
            [ IBM-DeskStar-25GP-10,1 GB | Seagate-Barracuda-9-9,1GB]
          private controller: controller-t
        }

        public cpu: {
          public name: [ Intel-Celeron-A-366MHz | Athlon-AMD-500 ],
          private slot: cpu-slot-t
        }

        rules
      motherboard.slot = cpu.slot,
      motherboard.controller = harddisk.controller,

      motherboard.name = Abit-BX6-ATX = >
        motherboard.slot = SLOT-1 /\ motherboard.controller = IDE,

      motherboard.name=Aopen-AX6BP-ATX =>
        motherboard.slot = SLOT-1 /\ motherboard.controller = SCSI,

      motherboard.name = Aopen-AK-72-KX133-ATX =>
        motherboard.slot = SLOT-A ^ motherboard.controller = IDE,
```

harddisk.name = IBM-DeskStar-25GP-10,1GB =>harddisk.controller =IDE,

harddisk.name = Seagate-Barracuda-9-9,1GB =>harddisk.controller = SCSI,

cpu.name = Intel-Celeron-A-366MHz =>cpu.slot = SLOT-1,

cpu.name = AMD-Athlon-500 =>cpu.slot = SLOT-A

第一部分是声明将用来定义产品变量类型的类型。接下来的部分是声明产品变量。每个变量都有一个标志符和一个类型。这个例子的类型系统包括原子结构、记录结构({...})和枚举类型(...|...| ...)。例如，cpu 就是一个包括由 name 和 slot 组成的记录的产品变量，且该 slot 是 cpu-slot-t.类型。cpu-slot-t 被声明为一个包括 SLOT-1 和 SLOT-A 两种替换件的枚举类型。public 和 private 指示符是用于控制在配置中，哪些部件或属性可以显示给最终用户（后面会给出具体描述）。第三部分声明规则。这些规则通常是产品变量的布尔公式，并且所有的规则必须满足兼容配置。通常，这些规则可以表示产品变量之间的任何关系，但在这个例子具体表示的规则可以分成两个不同的种类：

属性规则 为一个特定替换件的指定一个特定属性的值。例如，指定 Aopen-AX6BP-ATX 主板的 slot 的类型为 SLOT-1。

兼容性规则 指定不同部件的替换件/属性之间的一般相互关联性。例如，指定硬盘的控制器类型必须和主板的控制器类型相同。

在这个方法中，一个配置包括为产品变量的所有公有部分选定一个具体的值。在计算机的例子里，这包括选择选择主板的 name，CPU 的 name 和硬盘的 name。计算机的一个兼容配置是满足计算机产品型号规则的一个配置。一个兼容配置的例子是，选定 motherboard.name 为 Abit-BX6-ATX，cpu.name 为 Intel-Celeron-A-366MHz 和 harddisk.name 为 IBM-Deskstar-25GP-10,1GB。

在上述例子中，产品型号被文本化表示。但是，本发明并不局限于那样的表示。实际上，一个产品型号的完整描述划分为多个描述。本发明的一方面，将产品说明和产品表结合起来，以获得完整的产品型号。产品说明通常用于通过定义部件及其属性的方法来获得产品的结构信息，产品表通常用来获得部件的具体替换件与属性的具体值。

这种方法允许有大量的产品数据表，这些表通常也很难转成产品型号用于计算机辅助配置。这种应用包括构建一个房地产销售商店，其中房子的潜在购买者看起来是在“配置”他自己房子的。现实中，他是利用表示所有兼容配置的下拉菜单来在如 10，000 所房子里选择。在这个例子里，属性包括价格范围，位置，车库，游泳池，房间数量，面积等。

举例的计算机产品型号可分为一个产品说明和三个产品表。该产品说明与最初的计算机产品型号包含有相同的部分，但去掉了属性规则：

```
types
    cpu-slot-t = [ SLOT-1 | SLOT-A],
    controller-t = [ IDE | SCSI]

variables
    public motherboard: {
    public name:
      [ Abit-BX6-ATX | Aopen-AX6BP-ATX | Aopen-AK-72-
KX133-ATX],
    private slot: cpu-slot-t,
    private controller: controller-t
   }

   public harddisk:{
```

```
  public name:
    [ IBM-DeskStar-25GP-10,1 GB | Seagate-Barracuda-9-9,1GB]
  private controller: controller-t
 }

public cpu: {
  public name: [ Intel-Celeron-A-366MHz | Athlon-AMD-500 ],
  private slot: cpu-slot-t
 }

  rules
motherboard.slot = cpu.slot,
motherboard.controller = harddisk.controller,
```

第一个表定义主板部件的属性：

motherboard.name	motherboard.slot	motherboard.controller
Abit-BX6-ATX	SLOT-1	IDE
Open-AX6BP-ATX	SLOT-1	SCSI
Aopen-AK-72-KX133-ATX	SLOT-A	IDE

第二个表定义硬盘部件的属性：

harddisk.name	harddisk.controller
IBM-DeskStar-25GP-10,1GB	IDE
Seagate-Barracuda-9-9,1GB	SCSI

第三个也是最后一个表定义 CPU 部件的属性：

cpu.name	cpu.slot
Intel-Celeron-A-366MHz	SLOT-1
Athlon-AMD-500	SLOT-A

通过以下方法，可以从一个产品说明和一组产品表中获得一个文本化产品型号：

• 对每个表，将表转换为一个规则，

• 把前步骤获得的规则增加到产品说明里。

该表利用将一个表看成是分散的正常形式的表达式的关键观测转化为一个规则。一个 n 行 m 列的表可以按如下方式转换：

• 在标注为 y_i 的列里的、内容为 x_j^i、i 行 j 列的单元被转换为一个原子规则 $y_j = x_j^i$

• 对在 n 行中的行 i，从本行单元里获得的所有原子规则组合在一起，形成一个次规则 $(y_1 = x_1^i \wedge \ldots \wedge y_m = x_m^i)$

• 所有的 n 个 次规则通过拆分这些次规则来合并成一个大的规则：$(y_1 = x_1^1 \wedge \cdots \wedge y_m = x_m^1) \vee \ldots \vee (y_1 = x_1^n \wedge \cdots \wedge y_m = x_m^n)$

• 该大规则就是转换为一个规则的表。

例如，计算机产品模式的最后一个表转换成：

rules

（cpu.name = Intel-Celeron-A-366MHz /\ cpu.slot=SLOT-1）\/

(cpu.name = Athlon-AMD-500 /\ cpu.slot=SLOT-A)

一个方便的扩展是增加表过滤器来将产品表中的值映射到产品说明中的值。一个上述过滤器的例子是，将产品表中的特定价格（如 $100，$223，$817）映射到产品说明中的价格水平里（如便宜，合理，昂贵）。这种映射，通常是将一个范围里的所有价格映射到同样的价格水平。

向规则的转换不需要显式地执行。本发明的一个方面是，转换可以在构建虚拟表的时候顺便完成。

产品型号的优选实施例

产品型号的优选实施例包括一个产品说明和一组产品表。产品说明是一个 XML 1.0 文档。产品表是 ODBC 数据源和 SQL 查询的组合。

如附件 A.10 所示，XML 文档是利用文档类型声明（DTD）来定义。一个产品说明基本包括：

常量声明 一个常量可以被显式指定（constant），或者指定为一个 SQL 查询，其在被估算时，返回一个单元 dbconstant。

类型声明 一个类型声明（type）声明一个类型标识符作为一个特定类型（参见如下）的简写。

产品变量 一个产品变量（productvariable）可以声明为一个公有或私有的给定类型（参见如下）。

规则 一个规则是满足配置兼容要求的产品变量上的布尔表达式。这个表达式可以显式指定（rule），或是利用一个在估算时应返回一个可转换为一个规则（dbrule）的表的 SQL 查询来指定。

数据库细节 最后，可以提供这样几个额外部分的信息：别名定义（Alias）定义一个 ODBC 数据源，SQL 查询定义（sqlqueries）和可用来在数据库中的值和产品说明的值之间映射的过滤器（filter）。

规则包括结构化表达式：诸如布尔表达式（真，假）的原子表达式，来自有界子范围（0,1,.....n）的值，由数组、记录表达式、和枚举（sum）表达式创建的复合表达式。而且，提供了算术操作符和布尔操作符。在该优选实施例中，允许的算术操作包括加法、减法、乘法，其中乘法操作只有在至少有一个操作数是常数的情况下，才能使用。一开始，算术操作允许的类型看起来奇怪，以后将看到，这个方

法和虚拟表的优选实施例一起很好地工作。

选择 XML 作为产品说明的语言为了说明，允许直接转换成文本化格式和在计算机上表示的树形数据结构。

开发产品说明的一个优选方法是利用图形用户界面。ConfigItStudio 就是这样的一个图形用户界面，参看图 2 中的屏幕图。该屏幕图显示在编辑一个 PC 产品型号时的 ConfigItStudio 产品型号编辑器。在该屏幕图左边的树形视图是产品说明的树形视图，它和 XML 文档类型声明密切相关。（在屏幕图中，术语“template”是用于类型说明，术语“constraint”是用于规则。）右边的区域显示树中选定顶点的细节并可用于操作该顶点。在屏幕图顶端的菜单（“Complie”菜单）可构建一个产品型号的虚拟表并为在互联网上的交互式配置运行一个虚拟表服务器（“Run”菜单）（见下文）。

将产品型号编码为一个虚拟表

本发明的一个重要方面是将产品型号转换为一个简洁、有效的表示。这个过程被成为虚拟制表，这个结果的表现形式被称为一个虚拟表。很多种方式可以用来完成这种转化。该转化的目的是首先查找一种编码的方法并找到配置问题中所有解决方案，然后将它们虚拟制成一个虚拟表，以便通过对虚拟表的有效查询就能获得涉及配置问题的信息。编码包括查找产品型号部件的编码和规则的相应编码。一个 DAG 将表示所有的规则，因此，可以有效地执行对规则的有效解决方案的查找。虚拟表包括这种 DAG 和涉及产品型号与 DAG 之间关系的信息。

本发明较之现有技术的好处来自利用一个 DAG 以下面的方式来表示所有规则的步骤，即查询可以有效进行，好像实际上存在一个所有解决方案的表一样。一个完全表通常都太大了而无法实用，然而适当挑选的编码可以在保证将所有解决方案归入表格又保持准确性的同时产生小的 DAG。

虚拟表最重要的部分是表示每个兼容的配置的 DAG。由于一个真实生活中的产品型号难以置信的有很多配置，因此 DAG 必须以某种方式隐式的捕捉到这些配置。而且，DAG 必须准确的表示这些配置（即是没有“丧失准确性”）。对 DAG 的需求分为以下两类：

功能需求 DAG 须能够表示一个配置集，其中的每个配置都为每个产品变量定义一个值。DAG 中的基本算法须模拟上述配置集中的操作和功能：

- 构建集联合和构建一群配置集的交集，创建建两个配置集之间的集差，以及改变、约束或扩展一个配置集中一个变量的可能值，等，
- 检测集空，集包含和集等价。确定一个变量的可能值和确定在一个配置集中配置的数量。

效率需求 产品型号中规则的性质/结构意味着所介绍的算法中的许多算法都有一个随着产品变量数量而（至少）指数级增长的、典型的最坏情况下的运行时间。DAG 的大小也在最坏情况下（至少）随着产品变量数量而指数级增加。然而，对于真实生活中的产品，算法应有效运行，DAG 表示应简洁。

这些需求一开始看起来好像很难实现，但是对于一个现实生活中的产品型号，这种 DAGs 是的确存在的。

一个布尔决策图（BDD）是一个其中每个节点都包含布尔变量的 DAG。在硬件电路形式验证的领域里公知，BDDs 可为下面的类型（其中 n 是布尔变量的数量）的任意布尔函数编码：

$$B^n \to B \quad 。$$

这些函数应和“布尔产品型号”的配置集同构。(“布尔产品型号”

是产品变量的值都限定为真与假的产品型号)。因此，如果能对如布尔产品型号之类的一般产品型号编码，而且，如果根据基本 BDD 操作可以表达需求配置算法，那么可以 1）利用 BDDs 可以表示一般产品型号的虚拟表，及 2）利用该虚拟表执行对一般产品的实际配置。

BDDs 可以满足所有这些需求，并且，对于大多数真实生活的产品型号，这些算法有效且 DAGs 简洁。事实上，BDDs 是 DAG 的优选实施例。

但是，本发明并不限于所述 DAGs。很多其他的 DAGs 都有可被分别看作为集与集上的操作的表示和算法。必须根据表示产品型号规则的语言来仔细选择 DAG。

例如，差额决策图（参见 Moller et al: Difference Decision Diagrams,In proceedingsAnnual Conference of the European Association for Computer Science Logic(CSL),September 20-25 1999, Madrid, Spain）可表示 type R→B 的函数（的子集），并同时提供所需算法。一个直接的好处是，有了产品型号的编码方法，在该产品型号中，规则包括连续域的变量之上的定量表达式（的有限子集）。另一方面，该算法的缺点是不够有效（可编码规则的满足性是 pspace-hard 问题）。

另一个有关方法是在产品型号规则包括很多一般算术操作时，利用解释布尔变量上的 BDDs（参见 W.Chan, R.J.Anderson, P.Beame 和 D..Notkin: Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints, In O.grumberg,editor,Computer Aided Verification,9th International conference,CAV'97 Proceeding,Volum 1254 of Lecture Notes in Computer Science,pages 316-327,Haifa,Israel,June 1997,Springer-Verlag.)。每个布尔变量表示一个公式，DAG 中的一条路径表示这些公式的结合和该路径可满足性的联合，利用如线性程序设计等方法可以确定一条路径。

下面将在一个优选实施例中说明如何（利用 BDDs）将产品型号编码为一个虚拟表。但是，该领域内的技术人员能利用不同的潜在结构数据应用该算法，例如以上提到的两种数据结构中的一种。

将产品型号编码为虚拟表的优选实施例

将产品型号编码为虚拟表的优选实施例包括以下步骤：

静态扩展　产品型号通过展平（flattening）类型层次结构来扩展。结果是一个展平的产品型号和一个将该产品型号与展平的产品型号相关联的符号表。

BDD 编码　为每个规则都构建一个 BDD，创建的一个大 BDD 表示所有兼容配置。

下面，我们首先说明如何执行静态扩展。展平的产品型号是这种静态扩展的结果，这种模型的创建使得它适合利用 BDDs 编码。

静态扩展

静态扩展是通过展平类型层次结构来完成。结果是一个展平的产品型号和一个将该产品型号与展平的产品型号相关联的符号表。

展平的产品型号通过下列方式获得：1）除去记录表达式，2）简化域，及 3）以布尔形式编码。对于包含记录类型的每个产品变量，利用一个展平变量的列表代替产品变量来去掉记录类型。而且，这种产品变量的表达式都要被展平变量的表达式代替。替换后，所有的记录都从产品型号中去除。对计算机的产品型号而言，这一步骤产生下列产品型号（回忆上文，motherboard 是一个包括 name,slot 和 controller 记录类型的产品变量）：

```
types
  cpu-slot-t = [ SLOT-1 | SLOT-A],
```

```
        controller-t = [IDE | SCSI ]

    variables
        public motherboard_name:
            [ Abit-BX6-ATX | Aopen-AX6BP-ATX | Aopen-AK-72-
KX133-ATX],
        private motherboard_slot: cpu-slot-t,
        private motherboard_controller: controller-t,
        public harddisk_name:
          [ IBM-DeskStar-25GP-10,1GB | Seagate-Barracuda-9-9,1GB],
        private harddisk_controller: controller-t
        public cpu_name: [ Intell-Celeron-A-366MHz | Athlon-AMD-
500 ],
        private cpu_slot: cpu-slot-t,

    rules
        motherboard_slot = cpu_slot,
        motherboard_controller = harddisk_controller,

        motherboard_name = Abix-BX6-ATX =>
          motherboard_slot = SLOT-1 ^ motherboard_controller = IDE,

        motherboard_name = Aopen-AX6BP-ATX =>
            motherboard_slot = SLOT-1 ^ motherboard_controller =
SCSI,

        motherboard = Aopen-AK-72-KX133-ATX =>
            motherboard_slot = SLOT-A ^ motherboard_controller =
IDE,
```

harddisk_name = IBM-DeskStar-25GP-10,1GB => harddisk_controller = IDE,

harddisk_name = Seagate-Barracuda-9-9,1GB => harddisk_controller = SCSI,

cpu_name = Intel-Celeron-A-366MHz =>cpu_slot = SLOT-1,

cpu_name = AMD-Athlon-500 => cpu_slot = SLOT – A

展平产品型号的第二步包括简化展平的变量的域。所有展平值都转换为数字，每个展平的变量的域都转换为一个区间。例如，在 cpu_slot 中，值 0 代替 SLOT-1(这是 cpu slot 的第一种替换件) ，值 1 代替 SLOT-A（第二种替换件）。在计算机产品型号里，最后得到的产品型号是：

```
variables
public motherboard_name : 0 ..2,
public harddisk_name: 0..1,
public cpu_name : 0..1,

private motherboard_slot : 0..1,
private cpu_slot: 0..1,

private motherboard_controller: 0..1,
private harddisk_controller: 0..1

rules
    motherboard_slot = cpu_slot,
motherboard_controller = harddisk_controller,
motherboard = 0 => motherboard_slot = 0 /\ motherboard_controller =0
motherboard = 1 =>motherboard_slot = 0 /\ motherboard_controller = 1,
motherboard = 2 => motherboard_slot =1 /\ motherboard_controller
```

=0,

harddisk = 0 =>harddisk_controller=0,

harddisk = 1=>harddisk_controller =1,

cpu=0 =>cpu_slot = 0,

cpu=1 =>cpu_slot = 1,

展平产品型号的最后一步包括用布尔形式对产品型号编码。每个展平的变量都被一个布尔变量列表替换，每个规则都被这些布尔变量上的一个新规则替换。

在域为类型 0...n 中的展平的变量被$\lfloor \log_2(n+1) \rfloor$布尔变量代替。N+1 个数值里每个布尔变量都有一个唯一的赋值。例如，为对展平的变量 motherboard_name 的 3一值域编码（n=2），需要两个布尔变量：X0 和 X1。这个域中的每个布尔变量都有唯一赋值：对于值 0:X0=0,X1=0,对于值 1：X0=0,X1=1,对于值 2：X0=1,X1=0。现在，每个规则都被该布尔变量上的新规则所代替，以获得展平的产品型号。例如，计算机产品型号的展平产品型号是：

variables

public X0,X1,X2,X3

private X4,X5,X6,X7

rules

X4=X5,

X6=X7

(X0=0 /\ X1=0)=>X4=0^X6=0,

(X0=0 /\ X1=1)=>X4=0^X6=1,

(X0=1 /\ X1=0)=>X4=1^X6=0,

X2=0=>X7=0,

X2=1=>X7=1,

X3=0=>X5=0.

X3=1=>X5=1

在产品型号的展平过程中构建了一个符号表。这个符号表包括两个表。第一个表包含类型的信息，每个展平的变量的域以及用来为该变量的值编码的布尔变量。对于计算机产品型号，这个表是：

展平的变量	类型	整数域	布尔变量
motherboard_name	public	0..2	X0,X1
harddisk_name	public	0..1	X2
cpu_name	public	0..1	X3
motherboard_slot	private	0..1	X4
cpu_slot	private	0..1	X5
motherboard_controller	private	0..1	X6
harddisk_controller	private	0..1	X7

第二个表涉及展平值，它们的整数值和唯一的布尔赋值。对于计算机产品型号，这个表是：

展平的变量	展平的值	整数值	布尔值
motherboard_name	Abit-BX6-ATX	0	0,0
motherboard_name	Aopen-AX6BP-ATX	1	0,1
motherboard_name	Aopen-AK-72-KX133-ATX	2	1,0
harddisk_name	IBM-DeskStar-25GP-10,1GB	0	0
harddisk_name	Seagate-Barracuda-9-9,1GB	1	1
cpu_name	Intel-Celeron-A-366MHz	0	0
cpu_name	Athlon-AMD-500	1	1
motherboard_slot	SLOT-1	0	0
motherboard_slot	SLOT-A	1	1
cpu_slot	SLOT-1	0	0
cpu_slot	SLOT-A	1	1

motherboard_controller	IDE	0	0
motherboard_controller	SCSI	1	1
harddisk_controller	IDE	0	0
harddisk_controller	SCSI	1	1

BDD 编码

现在执行 DAG 的创建。该优选实施例是一个（降序的）二进制决策图。

利用布尔决策图来表示布尔公式是公知的。布尔决策图的介绍可以参看[Cristoph Meinel & Thorsten Theobald: Algorithms and Data Structure In VLSI Design,Springer 1998]。我们将使用下面是 BDDs 的文本化表示：

- 0 表示终端 BDD 0（真），
- 1 表示终端 BDD 1（假），
- $(a \otimes b)$表示由操作符⊗表示的任何二进制布尔操作应用 a 和 b 得到的 BDD。
- $\exists x.a$ 表示通过从 BDD a 中对变量 x 作存在数量词操作（existentially quantifying out）获得的 BDD。
- $(x \rightarrow a,b)$ 是表示公式 if x then a,else b 的 BDD，该公式也可以由更简单的操作符表示，如 $(x \wedge a) \vee (-x \wedge b)$。

BDDs 有一个公知的图形表示。图 5 是这种表示的一个例子。该图是一个在两个变量 X0 和 X1 上的 BDD。变量选定的排序是 $X_0 \leq X_1$，且 BDD 表示公式：

$$X_0 \rightarrow ((X_1 \rightarrow 0,1),1) = (-X_0) \vee (-X_1) \quad 。$$

用于 BDDs 构建和分解的 BDDs 上的基本操作在算法 1 中有概述。

算法 MK 用于构建顶点，算法 APPLY 用于将一个操作作用于两个顶点，算法 EXISTS 用于构建一个 BDD 表示 BDD 中的一个变量的存在数量词，算法 VAR,LOW 和 HIGH 是分解 BDDs 的简单函数：VAR(u)返回和顶点 u 相关联的变量，LOW(u)返回与顶点 u 关联的低级分支（low-son）以及 HIGH(u)返回与顶点 u 相关的高级分支（high-son）。算法 FULLONESAT(u)计算一个新的 BDD v 实现 $v \to u$，以便 BDD u 有一个合适的赋值（一般，u 必须是可行的，即包括至少一个解决方案）。算法 ANYSAT(u)为 u 中的变量返回一个合适的赋值（u 又必须是可行的）。最后，算法 SATCOUNT(u)返回满足 u 的赋值的数量。

BDD 表示的构建是基于展平的产品型号。首先，选定布尔变量的一个合适次序。该次序的选择对构建的 BDDs 规模很重要。在优选实施例中，通过使表示同种展平的变量的布尔变量相邻来选定次序。一个用于计算机产品型号的合适次序是

$$X0 \le X1 \le X2 \le X3 \le X4 \le X5 \le X6 \le X7 。$$

选定一个次序后，将每个规则编码为一个 BDD。例如，第三规则（X0=0∧X1=0）=>(X4=0∧X6=0)的 BDD 通过对下面表达式编码（如图 4 所示）来创建：

$$(-X0 \wedge -X1) \to (-X4 \wedge -X6)$$

编码是利用公知的 MK 和 APPLY 算法实现。下面，集合 R 涉及到一组规则，其中每个规则被编码为一个 BDD。

然后，对每个展平的变量设定一个表示可能布尔变量赋值的域约束。例如，对展平的变量 motherbroard.name 而言，三种可能的值存在于（0，1 和 2）中。因此，两个布尔变量利用赋值（X0=0,X1=0）,(X0=0,X1=1)和(X0=1，X1=0)来对域编码。在图 5 中，显示了 motherboard.name 域约束的 BDD。注意余下的（未利用的）赋值（X0=1,X1=1）导向终端 0。由于所有其他展平的变量的域大小为 2

（与单个布尔变量相对应），所有的其他域约束 BDDs 被终端 BDD1 表示。创建这些 BDDs 的优选方法也是利用 MK 和 APPLY 算法。下面，集合 *D* 是指域约束集，其中每个域约束被编码为一个 BDD。

在这个阶段构建的 BDDs 将为产生一个表示所有规则 *R* 和所有域约束 *D* 的大 BDD 块而构建块。创建这个 BDD，首先将 BDDs 结合，使单个规则整合为一个 BDD R_{all}:

$$R_{all} \overset{def}{=} \bigwedge_{r \in R} r.$$

此处，$\bigwedge_{r \in R} r.$表示将 R 的所有元素 r 结合的结果。

图 6 显示了计算机产品的该 BDD。注意在该 BDD 中可能选择 X0=1 和 X1=1 且仍然达到 1 终端。这是因为未考虑域约束。因此建立包含所有域约束 D_{all} 的 BDD：

$$D_{all} \overset{def}{=} \bigwedge_{d \in D} d.$$

一个表示所有可能的相容性配置的 BDD 通过合并所有规则与所有域约束的所有 BDDs 来获得。

$$C_{all} \overset{def}{=} R_{all} \wedge D_{all}$$

图 7 显示了计算机产品型号的这个 BDD。可观察到其中有三条包含所有导向终端 1 的变量的路径。因此，这个计算机产品型号恰好有三个兼容的产品配置（每个配置都基于不同的主板）。

构建 R_{all},D_{all} 和 C_{all} 的优选方法是利用算法 2 中显示的算法 MULTIAPPLY。给定一个结合的且可换的操作符 ⊗ 和一组顶点 $U=\{u_1,.....,u_n\}$，该算法返回一个 BDD 表示：

$u_1 \otimes \ldots \otimes u_n$ 。

如前所述，在配置中，最终用户只能利用公有展平的变量。可以仅在这些变量上构建一个更小的 BDD 而不丢失必要的信息。通过对私有展平的变量集（称为 V_{priv} 集）作存在数量词操作来构建该 BDD，优选利用算法 3 中的 MULTIEXSTS 算法（V^{B}_{priv} 指表示 V_{priv} 的布尔变量集）：

$$C_{pub} \overset{def}{=} \exists V^{B}_{priv} \cdot C_{all}$$

其中在变量 W=$\{x_1,\ldots,x^m\}$有限集上，利用$\exists W$作为 m 个量词：$\exists x_1,\ldots\exists x_m$的简写。

对于计算机产品型号，结果是图 8 中所示的 BDD。在该图中，从顶端顶点到终端顶点 1 的任何路径都表示一个或多个满足该计算机兼容性配置的一个或多个布尔变量赋值。如果一些变量没有出现在一条路径上，就需要表示更多的赋值：这些值可以是 0 或 1，且该结果仍能满足兼容性配置。利用符号表，可以将这些信息和原始的展平的变量关联起来。

早期量词操作（Early Quantification）

对于大的产品型号，为整个兼容性配置集初次构建 BDD 和之后的对私有变量作量词操作在构建过程中会产生很大的 BDDs。将已知的早期量词技术应用到编码过程中可以有更多的优势（参见[J.R.Burch, E.M.Clarke, D.E.Long: Symbolic Model Checking with Partitioned Transition Relations, Proceeding of the 1991 International Conference on VLSI]）。要注意的关键是，如果一个变量在一个规则中不是自由的，可以在规则的逻辑乘组合中“移动”存在数量词到逻辑乘运算之后：

$$\exists x.(a \wedge b) = a \wedge (\exists x.b) \text{ if x is not free in } a. \qquad (1)$$

适用该技术的优选实施例如下所述：构建一个 graph(V,E)，包括

顶点 V 和边 E，其中每个顶点标有一个展平的变量，每条边标有一个规则。该图是无向的，但是多条边可以连接两个顶点（一个复图）。该图包括：

• 每个私有展平的变量 v 的标有 v 的顶点。（该顶点被称为顶点 v）

•如果展平的变量 v 和 w 在规则 r 中都是自由的，对每对顶点 v,w 和规则 r，对应有一条边在 v 和 w 之间。（该边被称为 edge(v,r,w)）

一个强连通部件图基于这个结构图被建立（如参见：[Cormen, Leiserson, Rivest: Introduction to Algorithms,p.488-490]）。S 表示该强连通部件集。每个强连通部件包括一个子图。$G_i=(V_i, E_i)$表示这些子图中的第 i 个（$1 \le i \le |S|$）。V_i 是该子图中的私有展平的变量，$R_i \stackrel{def}{=} \{r|(v,r,w) \in E_i\}$是该子图中的规则。

现在，图 G_i 一个私有展平的变量在非 G_i 图中的所有规则里，都不是自由的。利用等式 1 中显示的内容，这意味着（V_i^B是表示 V_i 的布尔变量）：

$$C_{pub} = \bigwedge_{i \in \{1,\ldots|S|\}} (\exists V_i^B . \left(\bigwedge_{r \in R_i} (D_{all} \wedge r) \right))$$

通过执行下列步骤来获得规则和展平的变量的次序：

• 为每个子图 G_i 选定一个该子图固有的展平的变量 V_i 的次序。O_i 表示这些展平的变量的有序列表。

• 定义一个所有展平的变量的有序列表 $O \stackrel{def}{=} (O_1 \wedge \ldots \wedge O_n) = \langle v_1, \ldots v_{|V|} \rangle$

• 基于次序 O 来对规则排序。

最后一个步骤优选由如算法 4 中显示的算法 ORDERRULES 执行。该算法作为输入 1）有序展平的变量 O 的列表和 2）边 E。调用 ORDERRULES(O,E)返回一列规则集 $F = \langle F_1, \ldots F_{|V|} \rangle$，其中：

$$\forall i,j \in \{1,\ldots,|V|\}:(i<j \Rightarrow (v_i \cap free\,vars(Fj)=\Phi)) \qquad (2)$$

现将公式 1 和公式 2 结合起来，可以确定兼容配置集 C_{pub}，通过：1）对域约束 D_{all}，从一个 BDD 开始，2）反复地，随着增加 $i(1 \le i \le |V|)$，在该 BDD 上，首先将 F_i 中的规则结合，然后对表示 v_i 的布尔变量作量词操作。这是通过在算法 5 中显示的优选执行算法 CONJOINEXISTS 来执行的。该兼容配置集是（其中，O^B 是用于 $O,\langle v_1^B,\ldots,v_{|V|}^B\rangle$ 编码的布尔变量向量列表）：

$$C_{pub} = CONJOINEXSITS(O^B,F,D_{all})$$

在多数的 BDD 组件中， 初始化时就给定了声明变量的数量，比如 n。然后变量利用在 0 到 n-1 之间的索引来引用。利用声明变量的数量用于例如 SATCOUNT 算法中。尽管 C_{pub} 的自由变量只在公有布尔变量里，也不能仅仅利用 SATCOUNT 来计算兼容配置的数量。给定一个因数 2^j，其中 j 是私有布尔变量的数量。为了得到兼容状态的正确数量，可以将上述结果除以该因数，或重新初始化一个变量声明数量和公有布尔变量相同的 BDD 组件（然后需要根据新的变量索引来对 BDDs 重新编码）。这里选择后一种方法。

算术表达式编码

计算机产品型号不包括任何算术表达式。但是，如前描述的，产品型号的优选实施例确实允许某些小心选择的算术表达式：两个表达式的加法，两个表达式的减法和表达式和常量的乘法。这些算术操作被允许，因为 1）它们在产品定型的过程中是有用的，和 2）同时，存在这些表达式的有效 BDD 操作。

关键要注意的是，在静态扩展过程中，产品型号用布尔形式编码前，在所有规则里的所有表达式是这些基本算法操作的布尔组合，以及展平的变量的（不）等式。利用标准布尔等式，所有的规则可以按

下列语法产生的格式写（按 BNF 格式写）：

bexpr ::= *bexpr* ∧ *bexpr* （逻辑乘）

| -*bexpr* (取否)

| *aexpr bop aexpr* (算术操作符)

bop ::= <|≤|=|≥|>|≠ (布尔操作符)

aexpr ::=*aexpr aop aexpr* (算术操作符)

| *constant* |*variable* (原子算法表达式)

aop ::= + | - |* , (算术操作符)

其中，*constant* 表示一个常量，*variable* 表示一个展平的变量。

如何对 BDDs 中的算法操作编码是已经公知的。两份参考是[Alan John Hu:Techniques for Efficient Formal Verification Using Binary Decision Diagrams,Ph.D thesis,Stanford University,Department of Computer Science,Technical Report Number CS-TR-95-156]和[Jorn Bo Lind-Nielsen:Verification of Large Sate/Event Systems.Ph.D Thesis,Department of Information Technology,Technical University of Copenhagen,IT-TR:2000-032.]

产品表编码

如前所述，一个产品表（典型的是在一个数据库里表示）能用来充分地表示一条规则。不需要一开始就清楚的将产品表转换为一条文本化规则，然后再将规则转换为一个 BDD。实际上，可以直接从产品表里创建一个 BDD。

这个过程的优选实施例是算法 6 中的算法 VIRTUALIZETABLE。该算法将每个单元制成表格，为该单元创建一个 BDD 并在临时 BDD 节点中积累制表的结果。辅助函数 VIRTUALIZECELL 为一个特定的单元创建一个 BDD。实现该算法将利用符号表来找出如何将展平的变

量映射到布尔变量。

通过增加和表过滤器相关的信息到符号表里和改变辅助功能 VIRTUALIZECELL 以利用该信息，可以很容易的增加表过滤器。

组合类型编码

计算机产品型号包含枚举类型的值（如 [IDE | SCSI]）。 组合（sum）类型（在许多经典的类型体系中已知的）的类型是可以利用 BDDs 编码的更通常的类型。这种复合类型允许一个标记（如枚举类型中）和一个值（可以是任何类型）。一个可有（特定类型）可无的额外硬盘的组合类型例子是：

variables:

extraharddisk: [ABSENT | PRESENT of [IDE | SCSI]]

extraharddisk 的可能值是 ABSENT， PRESENT(IDE) 和 PRESENT(SCSI)。如需要对其中的一个值编码，必须 1）捕获指明是选择 ABSENT 还是 PRESENT 的标记和 2）在后种情况下，选定了哪种子值（IDE 或 SCSI）。优选的实施例是对这两部分各自编码。在这个特定的例子中，一个布尔值（P）指明额外硬盘是 PRESENT，一个布尔变量（T）指明该类型是 SCSI。

在 P=false 的情况下，对值 T 进行编码没有意义。（在 P=false 时，有两种不同的值分配：P=false,T=false 和 P=false,T=true）。为了限制表示的大小并能够统计有意义的赋值数量，需要对每个子值选定一个默认值。然后，定义一个规格化的约束表示当没有选定子值时，子值有默认值。在 extraharddisk 例子中，选定 T 的默认值为 false，规格化约束是：

（P=false）->(T=false)

出现这种组合类型，所有规格化约束必须在 BDD C_{pub} 中结合，

以获得表示兼容配置集的最终 BDD。

利用虚拟表的交互式配置

现利用虚拟表来执行一个配置。不作为对本发明的限制，用户通常参与该过程。用户同一个与虚拟表相连的计算机程序——即配置助理——交互。一般，用户执行的配置会话（session）和计算机程序可以被看作是一个交互式的反复配置，其中配置助理在配置过程中引导用户：

- 配置助理利用虚拟表寻找提供给用户的信息，及
- 用户提供涉及他/她想法的信息。

有多种方法创建可以应用于该配置对话的“协议”。不作为对本发明的限制，一个对话可以典型的是一个包括下列步骤的反复过程（从配置助理角度看）：

1．检查虚拟表。 虚拟表中的信息量巨大。因此，配置助理必须能从虚拟表中抽取有限的信息。在给定的情况下（通常是在先选择），提供的信息必须仍是充分和相关的。

2．将该信息提供给用户。 必须以一种方式提供这种信息，以便用户能分辨在给定的时间里他/她有哪些可选项，以及这些选择如何影响配置产品的相容性。

3．允许用户选定/取消选择。 在本阶段，配置助理已提供信息给用户，使得用户能容易的完成兼容的选择，但是在这种情况下，仍有可能用户做出和之前选择不兼容的选择。配置助理必须以一种合理的方式来处理所有这种类型的情形。怎样算合理取决于具体的应用。但是，通常配置助理需要牺牲一些之前的选择，然后达到一个兼容的选择集（将这些牺牲告知用户）。

4．利用虚拟表计算用户作出选择的后果。

这种反复过程一直持续到用户确定终止会话。如果此时找到了一个相容、完整的配置，那么可以将该配置提供给一个次序安排系统等。

用户和配置助理之间的通讯通过用户界面来执行。图 3 是一个用于 pc 产品型号用户界面的屏幕图。该用户界面允许用户查看已完成的选择(如,用户已为部件 Harddisk 1 选择了替换件 IBM DeskStar 25GP 10,1GB)，查看可用的替换件（这里，通过"popup"窗口，当前显示 cpu 的替换件）和这些替换件中的哪个替换件同其他之前选择相互兼容（这里，使用黑色的背景颜色）。用户可以为一个部件选择替换件，取消选择之前作出的配置，等。两个附加功能使用户更易操作：

• 一个按钮（ConfigIT!），用于当用户执行完他/她想要的选择时，让配置助理来完成配置（通过执行所有未选择部件的兼容性选择）

• 允许用户选择预配置选择的其他按钮，例如 pc 商店里的标准工作站。用户仍然可以自由的修改这个工作站，并能从配置助理处得到帮助。

一个通常的情形是在 web 浏览器里显示这样的一个用户界面，显然允许用户通过互联网通讯。虚拟表可以放在虚拟表服务器端或是运行 web 浏览器的客户端。在前一种情况下，配置助理包括一个服务器和一个客户线程（并发运行，通过互联网通讯）。在第二种情况下，配置助理典型地单独运行在 web 浏览器里。两种方法都是可行的，部署方法必须基于给定的配置会话对哪些属性有要求。但是，第一种方法是优选实施例。

在本结构框架里，部件和展平的产品变量对应，替换件和值对应。不作为对本发明的限制，下面的伪代码描述了配置助理通常是如何工作的（显然，细节可以用多种方法处理：用户可用的具体指令可以不同，来自系统的反馈可以不同，许多包括的步骤次序可以改变)：

$S \leftarrow \langle\ \rangle$

repeat

SHOWSTATUS(S)

$C \leftarrow READFROMUSER(S)$

```
        if C=exit then
            return S
        end
        S ← UPDATESELECTION(S,C)
    end
```

变量 S 是有序的选择列表，每个选择包括一对（v,d），其中 v 是一个展平的变量，d 是一个展平的值。该选择通常表示虚拟表中可用兼容配置的非空集。

一开始，一个选择也没有作，因此 S 是一个在虚拟表中表示完全兼容配置集的空列表。

可以从 S 中获得的信息和虚拟表被呈现给用户。典型的情况是，对每个展平的变量，只有和 S 兼容的可能选择才呈现给用户。

现在，向用户询问指令 C。该指令可以是一个同 S 相兼容的选择之一，取消 S 中已有的一个选择，强制选择，或当一个选择是为所有公有展平变量做的选择时，退出配置。

基于指令 C 和之前的选择 S，按照如下的方式，产生一个新的选择列表：

• 对于选择的情形，新的选择和 S 相兼容。新的选择仅仅是增加至 S 的末端。S 现在表示一个更小（或同等的）配置集。

• 对于取消一个之前选择的情形，是从列表 S 中简单的去掉该选择。S 现在表示一个更大（或同等的）配置集。

• 一个选择的强制情形有一些复杂。强制意味者“尽管该选择同其他的选择不兼容，还是强制执行这个选择，牺牲 S 里的其他选择。”通过下列步骤找出一个新 S’：

1. 在 S 的前端增加该新选择。（S 是一个有序列表）

2. 将 S'初始化为一个空列表〈 〉.

3. 从 S 的前端开始，对 S 中的每个选择 s：

一如 s 与 S'中的选择相兼容，则将 s 增加至 S'的末端。

一如 s 与 S'中的选择不兼容，则丢弃 s。

4. 新的选择就是 S'。

• 在退出情况下，结束操作并返回相容、完整的配置 S，并且例如传递到一个次序安排系统。

即使在用户和配置助理之间有关通讯的细节发生变化，在虚拟表上，实现伪代码的基本算法都大致相同。以下是通常所需的关键算法：

• 用于将虚拟表和一个或多个选择相结合以及检测这个结合是否相兼容的算法。

• 作用于一个虚拟表和一个可能不兼容配置的有序列表上的一个算法，它可以确定允许哪些选择，“牺牲”哪些选择来保证选择的兼容性。

• 作用于一个虚拟表和一些兼容性选择，为所有未选择部件确定兼容选择的算法。

• 一个用于计算给定选择集的兼容配置数量的算法。

• 作用于一个变量，确定和之前选择相兼容的可能选择的算法。

这些算法一般利用虚拟表中 DAG 的基本算法来实现（对于 BDD，算法 1 显示了该算法）。

交互式配置的优选实施例

交互式配置系统的优选实施例是一个包含分别运行在虚拟表服务器和 web 浏览器中的服务器和客户端线程的配置助理。首先，提供虚拟表上关键算法的优选实施例。然后，提供服务器和客户端算法。

基本 BDD 算法利用已知的一种称为动态编程的编程技术。因此，缓存 BDDs 的计算结果（取决于可用存储量，等）。这意味着可以在

一个恒定的时间里，运行带同样参数的再计算。许多算法利用：不是用临时 BDDs 来维护表，而是调用带同样参数的潜在的操作，并在一个恒定的时间里通常可以获得该结果。这种方法还可以在一个性能差一点（state-less）的服务器上实现。

最初的四个关键算法涉及到虚拟表和选择的结合。

在算法 7 中显示的第一个算法 CONFIG1 用于将一个选择和虚拟表（C_{actual}）合并。该选择包括一个展平的变量 v 和一个值 d。首先创建一个表示这个选择的 BDD。创建该 BDD 通过 1）确定表示 v 的布尔变量$\langle v_1,\ldots,v_n\rangle$和表示 d 的布尔值$\langle d_1,\ldots,d_n\rangle$（通过查询符号表），2）对包括布尔变量和布尔值的每对（v_i,d_i）为(v_i=d_i)创建一个 BDD，3）利用 MULTIAPPLY(→,·)将这些 BDD 结合获得一个 BDD。然后，u 和表示虚拟表的 BDD 结合产生一个新的 BDD。该 BDD 是有关原始虚拟表（例如，·表示产品型号规则）和选择（v=d）的兼容配置集。应注意，空配置集由 BDD 0 表示。因此，如果结果 BDD 是 0，则该选择不与虚拟表兼容。

回想在前部分讨论的计算机产品型号的例子。在图 8 中显示了表示规则和域约束的 BDD。可以利用 CONFIG1 来确定表示硬盘为 Seagate-Barracuda-9-9,1GB 的相容性配置集的 BDD。该结果显示在图 9 中。注意，只有一条路径导向 1。这意味着通过选择硬盘，其他所有的部件都可以暗含的被选定。通过观察 36 页的系统表，很容易发觉，这和 Aopen-AX6BP-ATX 主板、Seagate-Barracuda-9-9,1GB 硬盘，及 Intel-Celeron-A-366MHzCpu 相符合。

算法 8 中显示的算法 CONFIGCONSISTENT 构建一个表示几个选择 S_{new} 的 BDD。该算法简单的反复应用 CONFIG1，产生一个表示配置集越来越小的 BDD。注意，如果该选择是不相容的，则 BDD 返回 0，且不可能观察到“何时”发生问题。

算法 9 中显示的算法 CONFIGCHECK 用来处理这个问题。在 S_{new} 中的选择的次序可用来区分选择的优先顺序。如之前的算法 CONFIG1 被应用到各个选择。只要这些选择是兼容的，就能把它们增加到兼容选择列表中。但是，如一个选择和之前的选择不相兼容，则“拒绝”该选择，保留之前的配置集。这些被拒绝的选择也被收集在一个列表里。当算法结束时，它返回一个表示非空配置集的 BDD。在（合理的）假设下，当初始虚拟表非空时，该算法同样返回一个表示非空配置集的 BDD。还返回相容的和被拒绝选择的列表。

在算法 10 中显示的下一个算法 CONFIGIT 用于 1）基于一个相容性选择集限制配置集，和 2）为剩余产品变量自动选择一个兼容的值。该算法从产生一个表示非空解集 C_{actual} 的 BDD 的算法 CONFIGCONSISTENT 开始。然后，利用已知的 BDD 算法 FULLONESAT。该算法创建一个包含来自 C_{actual} 的单个路径的 BDD。该 BDD 只表示一个配置。然后，利用已知的 BDD 算法 ANYSAT 来获得由该路径表示的布尔选择。通过“反向”利用符号表，确定一个产品选择。

下面两个算法用来查询一个虚拟表。

算法 11 中显示的算法 CONFIGCOUNT 计算一个虚拟表中兼容配置的数量。算法 SATCOUNT 用来确定这个数量。但是，用于返回该数字的三个重要细节方面是：

1. 在虚拟表编码的优选实施例中，域约束和虚拟表相结合。没有域约束，将会有产生一个获得错误配置数量的“非法”路径。

2. 在虚拟表编码的优选实施例中，规格化所有的组合类型。没有规格化，值 ABSENT 的变量将会有多于一个的布尔选择（见第 44 页的例子），获得错误的配置数量。

3. 在虚拟表编码的优选实施例中，在创建虚拟表后，BDD 组件

重新初始化，从而去掉私有布尔变量。在 BDD 组件没有重新初始化的情况，需要将从 SATCOUNT 中获得的数量除以 2^n，其中 n 是私有布尔变量的个数（通过检查符号表获得）。

在表示计算产品型号虚拟表的 BDD 上使用 CONFIGCOUNT 将返回计算机兼容配置的数量，3。也可以通过计算导向 1 的路径数量得到这个数字。

在算法 12 中显示的算法 DETERMINEDOMAIN 为一个指定的展平的变量 v_i 和一个虚拟表 C_{acutal} 确定能选择哪个可能值。该算法对除表示 v_i 的布尔变量 v_i^B 外所有布尔变量 X 进行存在数量词操作。这种存在数量词操作的结果是一个从配置角度看来表示一个虚拟表的 BDD。该虚拟表具有只包括一列的优点（具有 v_i 的可能值的列）。由于依靠虚拟表形态的产品变量和其他的变量（仅一列）不能有内部关联性，因此可以通过简单地在一个虚拟表中列出各个元素来确定域。这是通过 1）找出虚拟表中布尔变量 v_i^b 的赋值集，2）将这些值每一个都转换为展平的值。

一个解释实例是为计算机产品型号的初始虚拟表确定 motherboard_name 展平的变量的域。表示 motherboard_name 的布尔变量是 X0 和 X1。对所有其他布尔变量进行存在数量词操作产生一个图 10 所显示的 BDD（和图 5 结构等同）。这个 BBD 有三个布尔变量 X0 和 X1 的赋值：(0,0),(0,1),(1,0)。利用符号表，可以确定相应的展平的值。

上述算法可以实现多种不同的配置系统。现在说明这些算法如何在一个包括通过网络（如互联网）通讯的服务器和客户线程的配置助理上使用。虚拟表位于在配置中执行所需计算的虚拟表服务器上。一个客户端程序用于向用户呈现有关配置过程的信息及关于选择的查询等。

优选的实现包括两个线程 CONFIGCLIENT(位于客户端)和算法 CONFIGSERVER(位于服务器端)。在算法 13 和算法 14 中分别显示了这两个算法。

CONFIGCLIENT 线程运行在一个配置会话期间。当用户找到一个合适的配置，希望结束配置会话时，该算法返回获得的配置（“返回”在实现中，被发送结果到一个次序安排模块等代替）。

CONFIGSERVER 算法是永不终止的。一开始，它就进入一个循环等待客户服务。当从客户那里接收到一个配置指令时，它计算出一个结果并立即返回该结果给客户。然后它又循环到开始状态，等待从一个新客户（或可能是同一个客户）传来的一个新指令，等等。

通讯利用如 SEND 和 RECEIVE 的标准原语来指定。该协议是：一开始，客户端程序发送一个配置指令到服务器。该服务器接收到该配置指令并计算出结果。该结果返回给客户。客户端程序接收到结果，并提供信息给用户及向用户询问用户命令（一个选择，退出等）。接收到用户指令一个新的配置命令，客户端程序发送到服务器，等等。

一个参数提供给该客户端程序：公有展平的变量集。该算法如下所述运行：

1. 首先，一个配置命令被发送到该服务器，表示“提供有关空的选择列表的信息给我”。

2. 然后，从服务器接收信息，包括：

S_{actual} 用户已作出的一个选择列表，

$S_{rejected}$ 利用一个 FORCE 操作，获得的由于不兼容性而被拒绝的选择列表（细节如下）.

N 表示已作出选择的兼容配置数量

$\langle D_1,\ldots,D_n\rangle$ 每个公有展平的变量 v_i 的，不会导致不兼容

配置的可选择值集

3. 该信息提供给用户

4. 接收到用户的指令。可能的用户指令是：

• Select(v,d) 增加一个值为 d 的公有展平的变量 v 的选择（即，一对(v,d)）。用户界面必须保证 if v is the variable v_i,then d $\in D_i$（因而，该选择导向一个兼容配置）。

• Force(v,d) 增加一个值为 d 的公有展平的变量 v 的选择。如果该选择和一个之前的选择 s'不相兼容，用户希望拒绝 s'。

• Deselect(v) 去掉公有展平的变量 v 的选择。

• Reset. 去掉所有的选择。

• PreConfigure(S) 去掉所有的选择，并以一个选择 S 的标准列表代替（例如，一个标准的工组站）。

• ConfigIt 对于所有的公有展平的变量，如果一个值没有选定，则让配置程序挑选任何兼容的选择。

• Stop 退出交互配置。服务器返回实际选择给调用程序，如一个次序安排程序。用户界面必须保证当前的选择包括一个完全的选择（即 N 必须为 1）。

5. 现在如果用户指令是 Stop，那么该算法终止并返回所获得的选择。

6. 其他情形，创建一个配置指令并发送到服务器。可能的配置指令有：

• Config(S) 命令服务器计算有关选择 S（可能是不兼容的）的信息。通过以下步骤创建选择 S：

一对一个 Select 用户指令，新选择仅被简单地加到选择列表的末端。

一对一个 Force 用户指令，新选择被加入到选择列表的前端：如果存在不兼容，则将接受一个强制选择，并拒绝其他的选择（这些选择以后会放到选择列表里）。

一对一个 Deselect 用户指令，从选择列表中去掉相关的展平的变量。

一对一个 Reset 用户指令，新选择为空。

一对一个 PreConfigure 用户指令，新选择基于标准列表。

• ConfigIT 计算关于兼容性选择 S 的信息。对有一个变量没有选定值的所有的公有展平的变量，选定任何兼容性选择。

7. 然后，又从服务器中接收到信息等，参见步骤 2。

服务器算法的目的是为了满足客户算法的计算需要，并被看成是客户端和先前说明的关键配置算法之间的连接代码。如下参数被提供给服务器算法：

C_{basic} 表示虚拟表的 BDD C_{pub}。

$\langle v_1,\ldots,v_n\rangle$ 公有展平的变量列表

Y 符号表

在计算过程中，变量 S_{actual} 是一个用户作出的选择的有序列表。每个选择包括一对(v,d)，其中 v 是一个公有展平的变量，d 是来自 v 的域的值。BDD C_{actual} 表示关于实际选择 S_{actual} 的所有兼容配置。

该算法运行如下：

• 服务器接收一个配置指令和一个选择 S_{new} 的有序列表。

• 如果指令是 Config，则 S_{new} 可能是不兼容的。因此，算法 CONFIGCHECK 用于创建一个兼容性配置集，以及创建接受的选择和拒绝的选择的列表。

• 或者，如果指令是 ConfigIt，则 S_{new} 是兼容的，但是也必须作出剩余的选择。这通过利用 ConfigIt 指令来实现。

• 然后，确定兼容性配置的实际数量。

• 为每个变量，确定实际可能的值。

• 计算的信息被返回至服务器，算法循环等待新的指令。

最后，配置助理在互联网上部署的优选方法如下所述：

• 首先，一个希望执行配置的用户和 web 服务器相连。

• web 服务器返回要在用户浏览器中执行的 CONFIGCLIENT 的实现（优选用 Java Script 实现）。

• 当 CONFIGCLIENT 在客户端 web 浏览器初始化时，它和一个带虚拟表，运行 CONFIGSERVER 算法的虚拟表服务器（不必是 web 服务器）相连。

• 客户端和服务器端线程按如前所述方式通讯。

附录：产品说明的XML文档类型声明

```
<!-- DTD for ConfigIt projects - Copyright (C) 2000 ConfigIt -->
<!ELEMENT project (head, entities)>
<!ELEMENT head (name, description)>
<!ELEMENT entities ((constant | dbconstant)*, type*, productvariable*,
                                                (rule | dbrule)*, database?)>
  <!ELEMENT constant (name, description, expression)>
  <!ELEMENT dbconstant (name, description, sqlalias, filteralias, expression)>
  <!ELEMENT type (name, description, typeconstructor)>
    <!ELEMENT typeconstructor ((boolean | subrange | array | product |
                                sumtype | sumdb | idtype | label),
                               optional?)>
      <!ELEMENT boolean EMPTY>
      <!ELEMENT subrange (expression)>
      <!ELEMENT array (expression, typeconstructor)>
      <!ELEMENT product (prodvar*, (rule | dbrule)*)>
        <!ELEMENT prodvar ((private | public), name, description,
                           typeconstructor)>
          <!ELEMENT private EMPTY>
          <!ELEMENT public EMPTY>
      <!ELEMENT sumtype (sumvar*)>
        <!ELEMENT sumvar (name, description, typeconstructor)>
      <!ELEMENT sumdb (sqlalias, filteralias)>
      <!ELEMENT idtype (#PCDATA)>
      <!ELEMENT label EMPTY>
      <!ELEMENT optional EMPTY>
  <!ELEMENT productvariable (name, description, (private | public),
                             typeconstructor)>
  <!ELEMENT rule (name, description, expression)>
  <!ELEMENT database (alias | sqlquery | filter)*>
    <!ELEMENT alias (name, description, dsn, username, password)>
      <!ELEMENT dsn (#PCDATA)>
      <!ELEMENT username (#PCDATA)>
      <!ELEMENT password (#PCDATA)>
    <!ELEMENT sqlquery (name, description, query, dbalias)>
      <!ELEMENT query (#PCDATA)>
      <!ELEMENT dbalias (#PCDATA)>
    <!ELEMENT dbrule (name, description, sqlalias, mapping*)>
      <!ELEMENT sqlalias (#PCDATA)>
      <!ELEMENT mapping (variable, lambdaexpr, lambdavar, column, filteralias)>
        <!ELEMENT variable (expression)>
```

```
        <!ELEMENT lambdaexpr (expression)>
        <!ELEMENT lambdavar (#PCDATA)>
        <!ELEMENT column (#PCDATA)>
        <!ELEMENT filteralias (#PCDATA)>
    <!ELEMENT filter (name, description, filterfunction, settings)>
      <!ELEMENT filterfunction (#PCDATA)>
      <!ELEMENT settings (number | true | false | string | numbers |
                          booleans | strings)*>
        <!ELEMENT string (#PCDATA)>
        <!ELEMENT numbers (number)*>
        <!ELEMENT booleans (true | false)*>
        <!ELEMENT strings (string)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT expression (idconstructor | number | true | false | and | or |
  xor | imp | biimp | plus | minus | mult | lt | lteq | gr | greq | eq |
  neq | shftl | shftr | not | forall | exist | sum | prod | case | sumvalue |
  if)>
  <!ELEMENT idconstructor ((idname, indexlist?) | (idconstructor,
                                                   idconstructor))>
    <!ELEMENT idname (#PCDATA)>
    <!ELEMENT indexlist (expression+)>
  <!ELEMENT number (#PCDATA)>
  <!ELEMENT true EMPTY>
  <!ELEMENT false EMPTY>
  <!ELEMENT and (expression, expression)>
  <!ELEMENT or (expression, expression)>
  <!ELEMENT xor (expression, expression)>
  <!ELEMENT imp (expression, expression)>
  <!ELEMENT biimp (expression, expression)>
  <!ELEMENT plus (expression, expression)>
  <!ELEMENT minus (expression, expression)>
  <!ELEMENT mult (expression, expression)>
  <!ELEMENT lt (expression, expression)>
  <!ELEMENT lteq (expression, expression)>
  <!ELEMENT gr (expression, expression)>
  <!ELEMENT greq (expression, expression)>
  <!ELEMENT eq (expression, expression)>
  <!ELEMENT neq (expression, expression)>
  <!ELEMENT shftl (expression | (expression, expression))>
  <!ELEMENT shftr (expression | (expression, expression))>
  <!ELEMENT not (expression)>
```

```
<!ELEMENT forall (idname, range, expression)>
  <!ELEMENT range (expression, expression)>
<!ELEMENT exist (idname, range, expression)>
<!ELEMENT sum (idname, range, expression)>
<!ELEMENT prod (idname, range, expression)>
<!ELEMENT sumvalue (name, expression)>
<!ELEMENT if (test, then, else)>
  <!ELEMENT test (expression)>
  <!ELEMENT then (expression)>
  <!ELEMENT else (expression)>
<!ELEMENT case (idconstructor, pattern+, expression)>
  <!ELEMENT pattern (expression, expression)>
```

附录B：算法

算法 1

function MK(x, h, l) : **Var** $\times V \times V \to V$
return (A vertex representing $(x \to h, l)$.)

function VAR(u) : $V \to$ **Var**
return (If u represents $(v \to h, l)$, return v.)

function LOW(u) : $V \to V$
return (If u represents $(v \to h, l)$, return l.)

function HIGH(u) : $V \to V$
return (If u represents $(v \to h, l)$, return h.)

function APPLY$(\otimes, u_1, u_2)$: **Operator** $\times V \times V \to V$
return (A vertex representing $(u_1 \otimes u_2)$.)

function EXISTS(x, u) : **Var** $\times V \to V$
return (A vertex representing $(\exists x.u)$.)

function FULLONESAT(u) : $V \to V$ ▷ $u \neq 0$
return (A BDD v (fullfilling: $v \to u$) with exactly one satisfying assignment.)

function ANYSAT(u) : $V \to$ (**Var** $\times \mathbb{B}$)-**set** ▷ $u \neq 0$
return (An assignments satisfying u.)

function SATCOUNT(u) : $V \to \mathbb{Z}$
return (The number of value assignments for the BDD u.)

function MIN(V) : **Var-set** $\to$ **Var**
return (The variable v in V with lowest ordering: $(\forall v' \in (V - \{v\}) : v \lesssim v')$.)

function MAX(V) : **Var-set** $\to$ **Var**
return (The variable v in V with highest ordering: $(\forall v' \in (V - \{v\}) : v' \lesssim v)$.)

算法 2

```
function MULTIAPPLY(⊗, U) : Operator × V-set → V        ▷ ⊗ associative and commutative
  if U = {0} then
    return 0
  elsif U = {1} then
    return 1
  elsif U = {0, 1} then
    return 0 ⊗ 1
  else
    i = MIN({VAR(v) | v ∈ U})
    V = {u ∈ U | VAR(u) = i}
    U' = U − V
    L = {LOW(v) | v ∈ V}
    H = {HIGH(v) | v ∈ V}
    return MK(i, MULTIAPPLY(⊗, H ∪ U'), MULTIAPPLY(⊗, L ∪ U'))
  fi
```

算法 3

```
function MULTIEXISTS(X, u) : Var-set × V → V
  if u ∈ {0, 1} then
    return u
  elsif VAR(u) > MAX(X) then
    return u
  else
    h = MULTIEXISTS(X, HIGH(u))
    l = MULTIEXISTS(X, LOW(u))
    if VAR(u) ∈ X then
      return APPLY(∨, h, l)
    else
      return MK(VAR(u), h, l)
    fi
  fi
```

算法 4

```
function ORDERRULES(⟨v_1, ..., v_n⟩, E) :
                    FlatVar-list × (FlatVar × Rule × FlatVar)-set → (Rule)-set-list
  for i ← 1 to n do
      E_i = {(w_1, r, w_2) ∈ E | (w_1 = v_i ∨ w_2 = v_i)}
      E = E − E_i
      F_i = {r | (w_1, r, w_2) ∈ E_i}
  end
  return ⟨F_1, ..., F_n⟩
```

算法 5

function CONJOINEXISTS($\langle v_1^B, \ldots, v_n^B \rangle, \langle F_1, \ldots, F_n \rangle, u$) : (Var-set)-list × ($V$-set)-list × $V \to V$
▷ $\forall i, j \in \{1, \ldots, n\} : (i < j \Rightarrow (v_i \cap freevars(F_j) = \emptyset))$

for $i \leftarrow 1$ **to** n **do**
  $u = \text{MULTIAPPLY}(\wedge, \{u\} \cup F_i)$
  $u = \text{MULTIEXISTS}(v_i^B, u)$
end
return u

算法 6

function VIRTUALIZETABLE(T, Y) : **Table** × **SymbolTable** → V
  $u \leftarrow 0$
  foreach row i in T
    $v \leftarrow 1$
    foreach column j in T
      $w \leftarrow \text{VIRTUALIZECELL}(T, i, j, Y)$
      $v \leftarrow \text{APPLY}(\wedge, v, w)$
    end
    $u \leftarrow \text{APPLY}(\vee, u, v)$
  end
  return u

function VIRTUALIZECELL(T, i, j, Y) : **Table** × **Row** × **Column** × **SymbolTable** → V
  return (a BDD representing $x_j^i = y_j$, x_j^i is the cell at (row i, column j), y_j is the label of column j.)

算法 7

function CONFIG1($C_{\text{actual}}, (v, d), Y$) : V × (**FlatVar** × **FlatVal**) × **SymbolTable** → V
  $u \leftarrow$ (A BDD representing the selection $(v = d)$.)
  return $\text{APPLY}(\wedge, C_{\text{actual}}, u)$

算法 8

function CONFIGCONSISTENT($C_{\text{basic}}, S_{\text{new}}, Y$) :
  V × (**FlatVar** × **FlatVal**)-**list** × **SymbolTable** → V
  $C_{\text{actual}} \leftarrow C_{\text{basic}}$
  foreach $s \in S_{\text{new}}$
    $C_{\text{actual}} \leftarrow \text{CONFIG1}(C_{\text{actual}}, s, Y)$
  end
  return C_{actual}

算法 9

function CONFIGCHECK($C_{\text{basic}}, S_{\text{new}}, Y$) : $V \times$ (**FlatVar** $\times$ **FlatVal**)-**list** $\times$ **SymbolTable** $\rightarrow$
($V \times$ (**FlatVar** $\times$ **FlatVal**)-**list** $\times$ (**FlatVar** $\times$ **FlatVal**)-**list**)

$C_{\text{actual}} \leftarrow C_{\text{basic}}$
$S_{\text{actual}} \leftarrow \langle\rangle$
$S_{\text{rejected}} \leftarrow \langle\rangle$
foreach $s \in S_{\text{new}}$ ▷ Must be "in-order" traversal.
  if CONFIG1(S_{actual}, s, Y) $\neq 0$ **then**
    $C_{\text{actual}} \leftarrow$ CONFIG1(S_{actual}, s, Y)
    $S_{\text{actual}} \leftarrow S_{\text{actual}}\hat{}s$
  else
    $S_{\text{rejected}} \leftarrow S_{\text{rejected}}\hat{}s$
  fi
end
return ($C_{\text{actual}}, S_{\text{actual}}, S_{\text{rejected}}$)

算法 10

function CONFIGIT($C_{\text{basic}}, S_{\text{new}}, Y$) : $V \times$ (**FlatVar** $\times$ **FlatVal**)-**list** $\times$ **SymbolTable** $\rightarrow$
($V \times$ (**FlatVar** $\times$ **FlatVal**)-**list**)
▷ S_{new} must be consistent wrt. to $C_{C_{\text{basic}}}$.

$C_{\text{actual}} \leftarrow$ CONFIGCONSISTENT($C_{\text{basic}}, S_{\text{new}}, Y$)
▷ $C_{\text{actual}} \neq 0$
$C_{\text{actual}} \leftarrow$ FULLONESAT(C_{actual})
$S^B_{\text{actual}} \leftarrow$ ANYSAT(C_{actual})
$S_{\text{actual}} \leftarrow$ (S^B_{actual} translated to flattened variable selections by "backwards" use of Y.)
return ($C_{\text{actual}}, S_{\text{actual}}$)

算法 11

function CONFIGCOUNT(u) : $V \times$ **Symboltable** $\rightarrow \mathbb{Z}$
return SATCOUNT(u)

算法 12

function DETERMINEDOMAIN($C_{\text{actual}}, v_i, \{v_1, \ldots v_n\}, Y$) :
$V \times$ **FlatVar** $\times$ **FlatVar-set** $\times$ **SymbolTable** $\rightarrow$ **FlatVal-set**

$X \leftarrow v_1^B \cup \cdots \cup v_n^B$
$X \leftarrow X - v_i^B$
$u \leftarrow$ MULTIEXISTS(X, C_{actual})
$D^B \leftarrow$ (The set of assignments of the Boolean variables v_i^b in the BDD u.)
$D \leftarrow$ (D^B translated to flattened values by "backwards" use of the symbol table Y.)
return D

算法 13

```
function CONFIGCLIENT({v_1, ... v_n}) : FlatVar-set → (FlatVar × FlatVal)-set
  SEND(Config, ⟨⟩)
  repeat
    RECEIVE(S_actual, S_rejected, N, ⟨D_1, ..., D_n⟩)
    I ← SHOWSTATUSANDREADFROMUSER(S_actual, S_rejected, N, ⟨v_1, ... v_n⟩, ⟨D_1, ..., D_n⟩)
    if I = Select(v, d) then                    ▷ ∀i ∈ {1, ..., n}.((v = v_i) → (d ∈ D_i))
      SEND(Config, STRIPSELECTION(S_actual, v)^(v, d))
    elsif I = Force(v, d) then
      SEND(Config, (v, d)^STRIPSELECTION(S_actual, v))
    elsif I = Deselect(v) then
      SEND(Config, STRIPSELECTION(S_actual, v))
    elsif I = Reset then
      SEND(Config, ⟨⟩)
    elsif I = PreConfigure(S_new) then
      SEND(Config, S_new)
    elsif I = ConfigIt then
      SEND(ConfigIt, S_actual)
    else                                        ▷ The selection must be complete.
      return S_actual                           ▷ If we get here: I = Stop.
    fi
  end
```

(Unspecified functions: SHOWSTATUSANDREADFROMUSER, SEND, RECEIVE)

```
function STRIPSELECTION(S, v) :
        (FlatVar × FlatVal)-list × FlatVar → (FlatVar × FlatVal)-list
  return (S modified so that a possible selection relating to the variable v is removed.)
```

算法 14

```
function CONFIGSERVER(C_basic, {v_1, ... v_n}, Y) : V × FlatVar-set × SymbolTable → ?
  repeat
    RECEIVE(C, S_new)
    if C = Config then
      (C_actual, S_actual, S_rejected) ← CONFIGCHECK(C_basic, S_new, Y)
    else
      (C_actual, S_actual) ← CONFIGIT(C_basic, S_new, Y)      ▷ If we get here: C = ConfigIt.
      S_rejected ← ⟨⟩
    end
    N ← CONFIGCOUNT(C_actual)
    for i ← 1 to n do
      D_i ← DETERMINEDOMAIN(C_actual, v_i, {v_1, ... v_n}, Y)
    end
    SEND(S_actual, S_rejected, N, ⟨D_1, ..., D_n⟩)
  end
```

(Unspecified functions: SEND, RECEIVE)

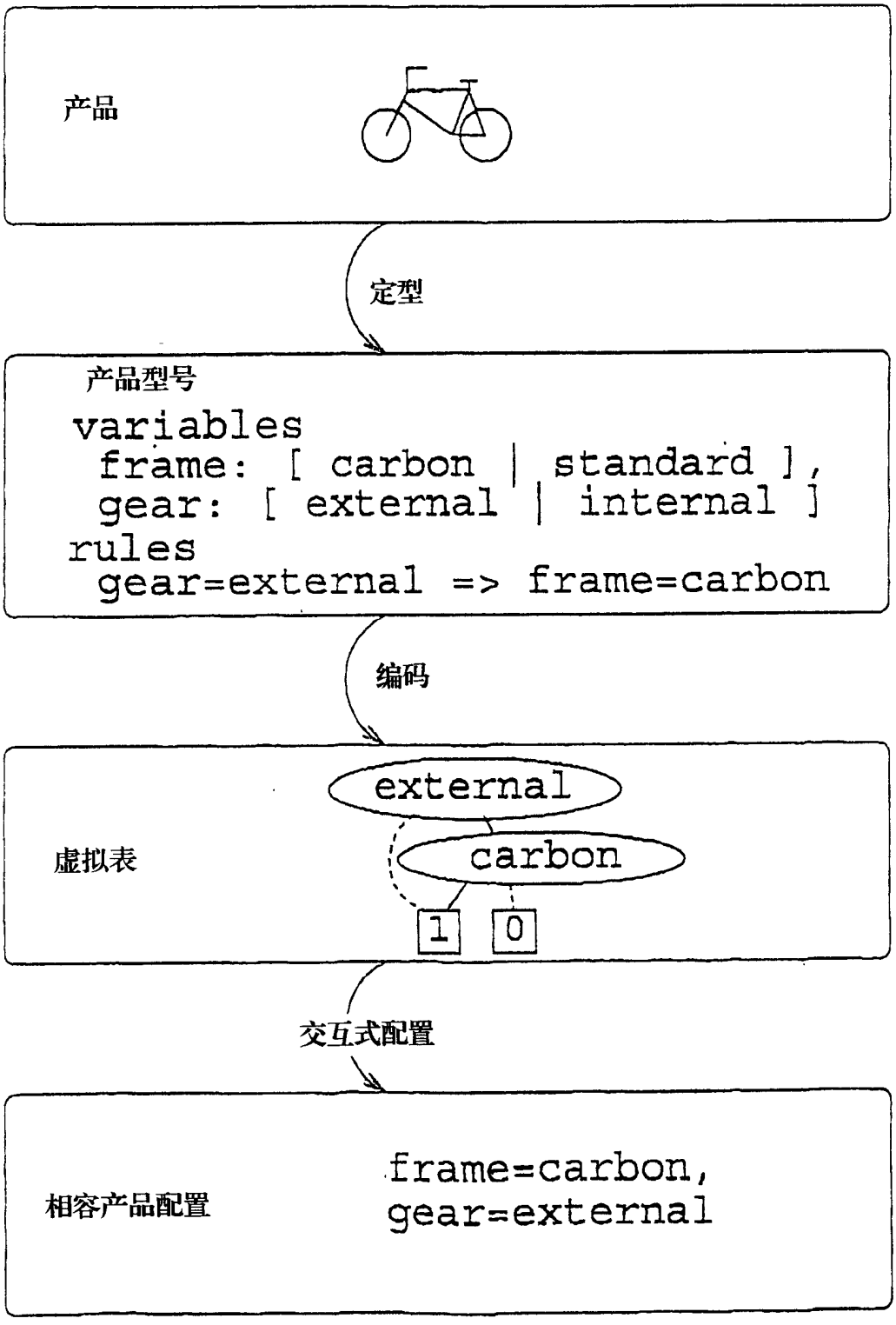

产品
定型
产品型号
variables
frame: [carbon | standard],
gear: [external | internal]
rules
gear=external => frame=carbon
编码
虚拟表
external
carbon
1
0
交互式配置
相容产品配置
frame=carbon,
gear=external

图 1

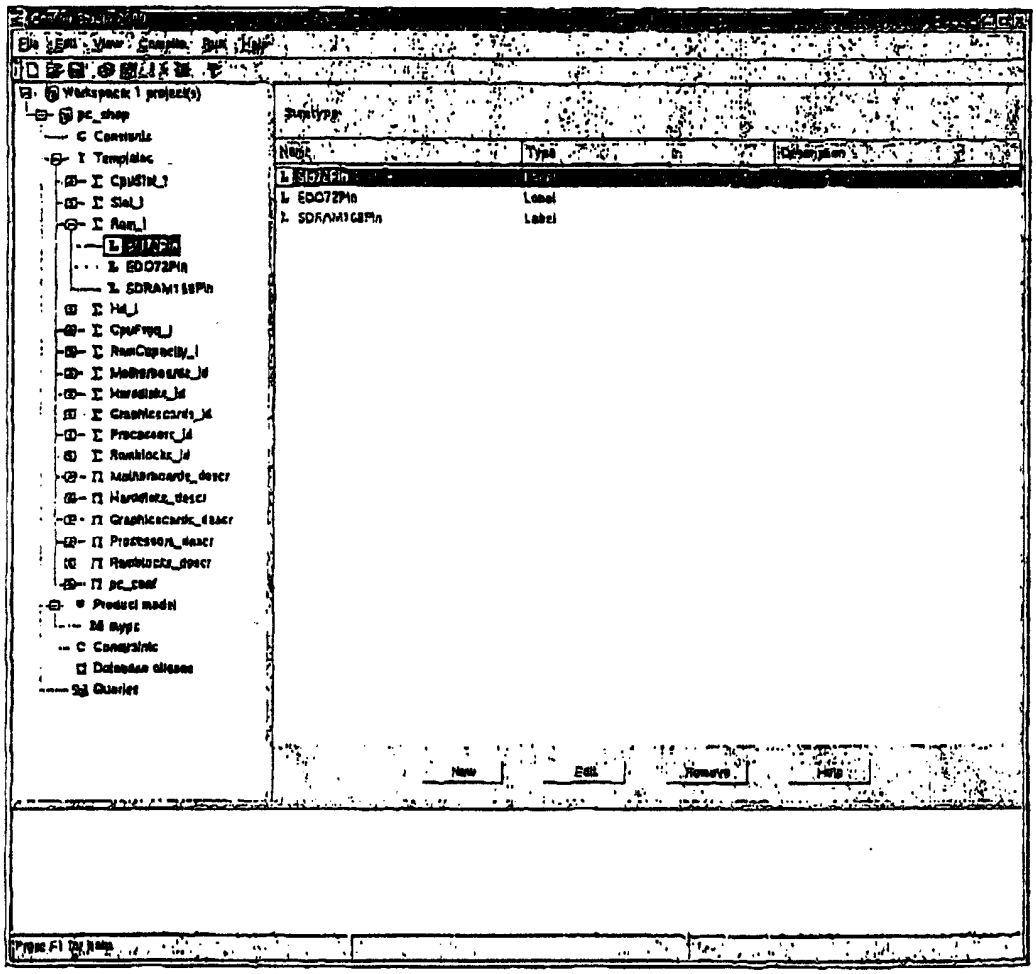

Workspace: 1 project(s)
pc_shop
Constants
Templates
Product model
Queries
Type
EDO72Pin
Label
New
Edit
Remove
Help

图 2

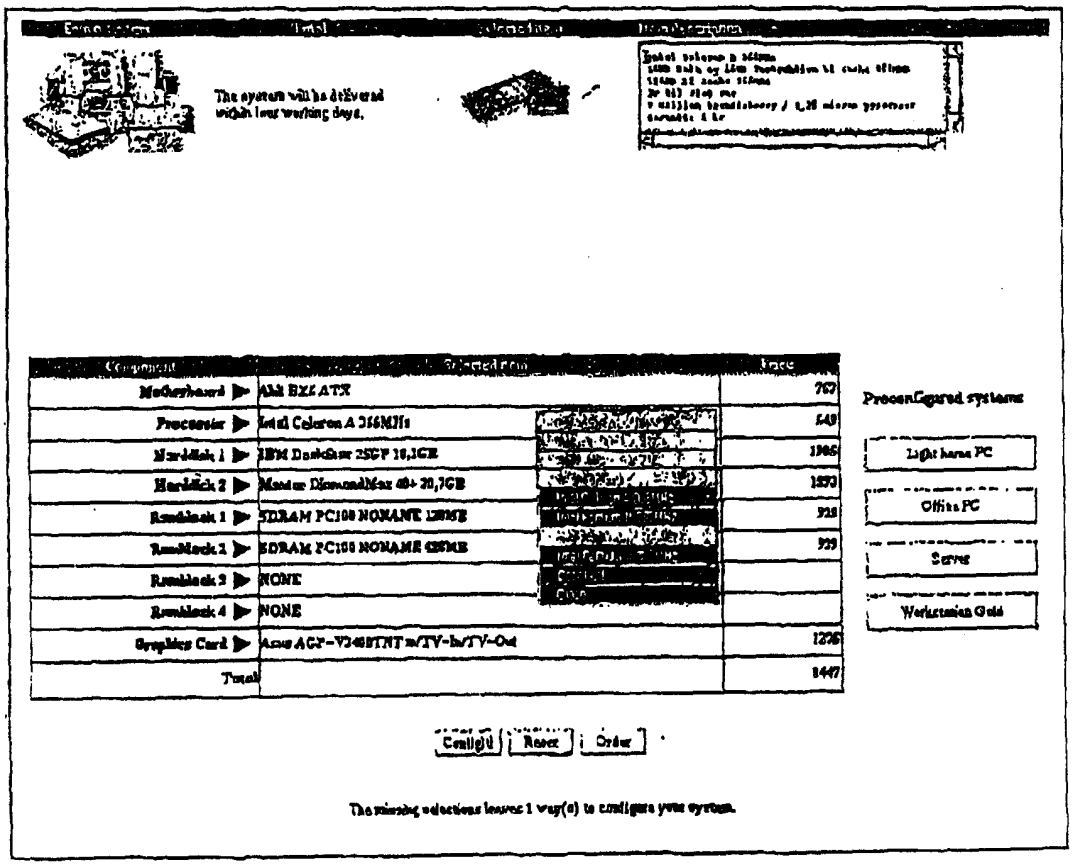

The system will be delivered
within four working days.
Light home PC
Office PC
Server
Workstation Gold
Config'it
Reset
Order

图 3

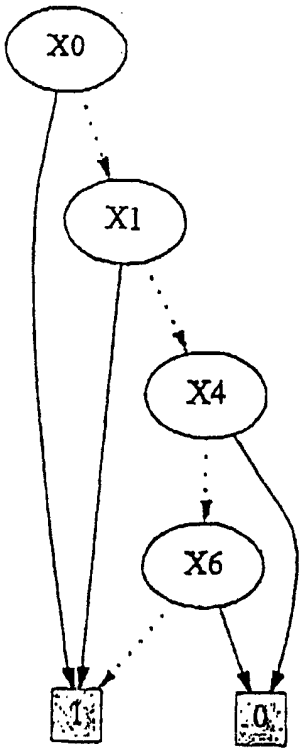


图 4

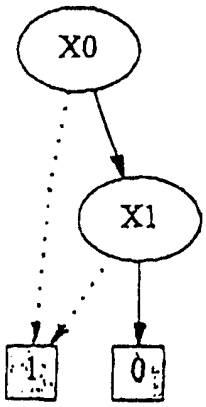


图 5

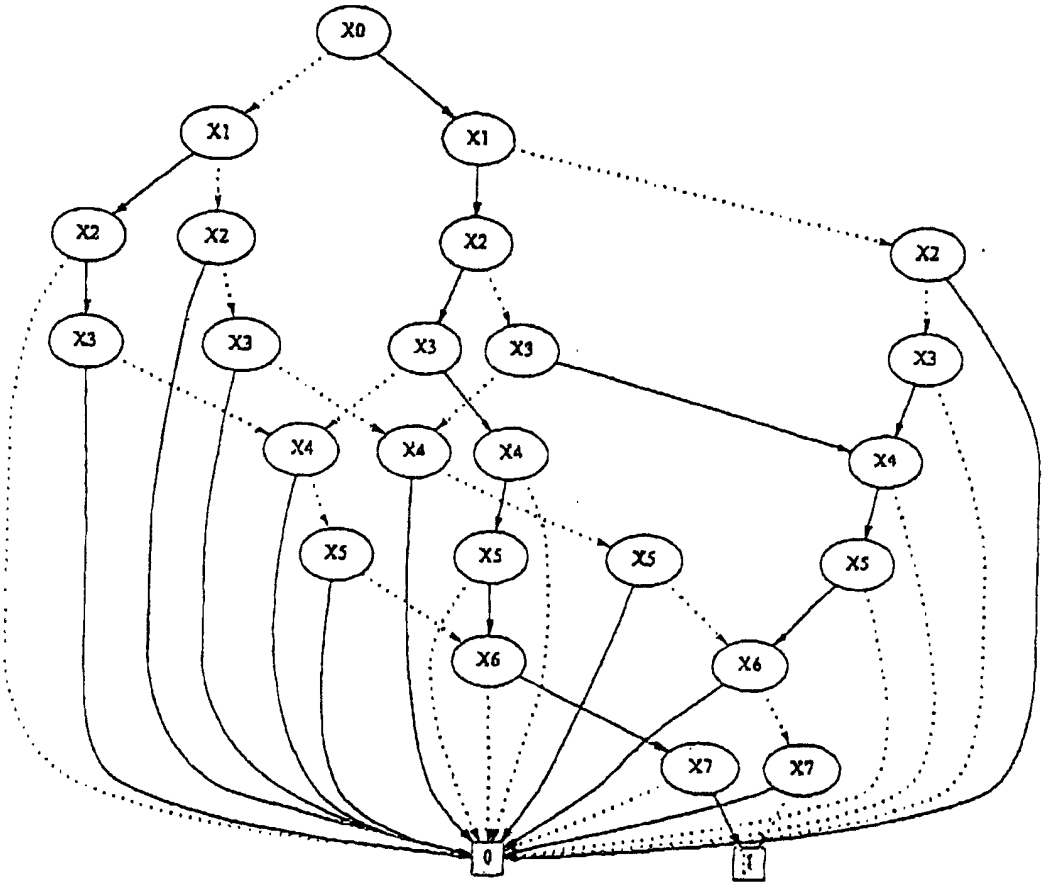

X0
X1
X1
X2
X2
X2
X2
X3
X3
X3
X3
X3
X4
X4
X4
X4
X5
X5
X5
X5
X6
X6
X7
X7
0
1

图 6

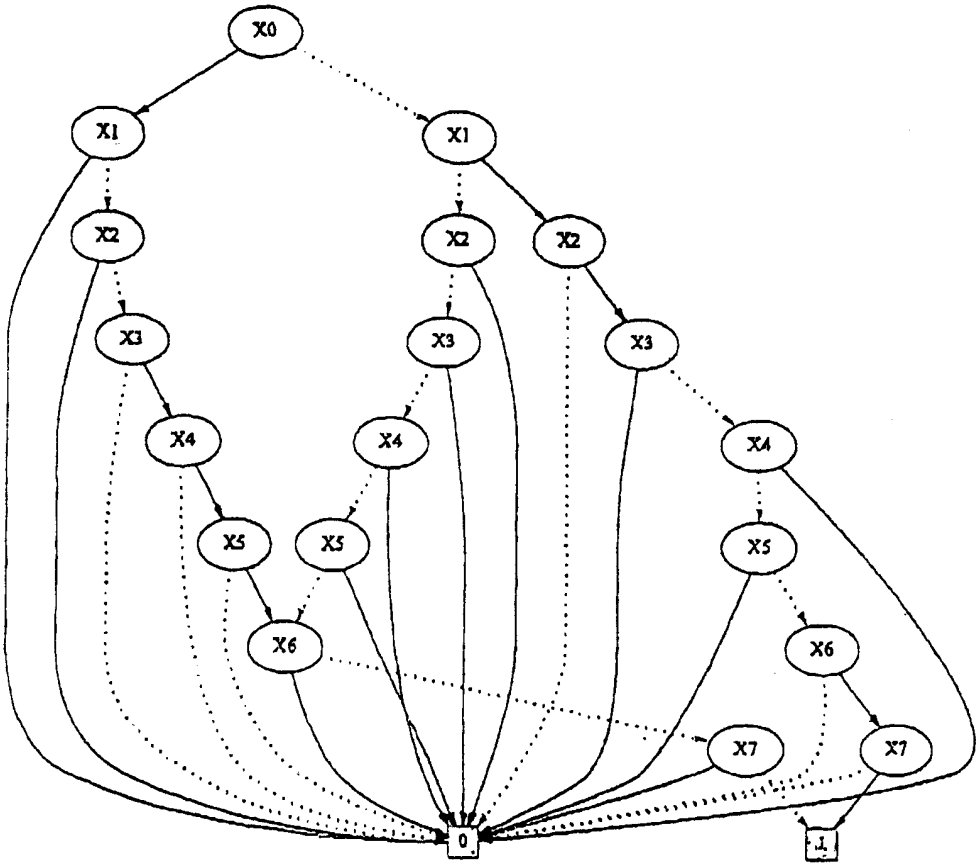


图 7

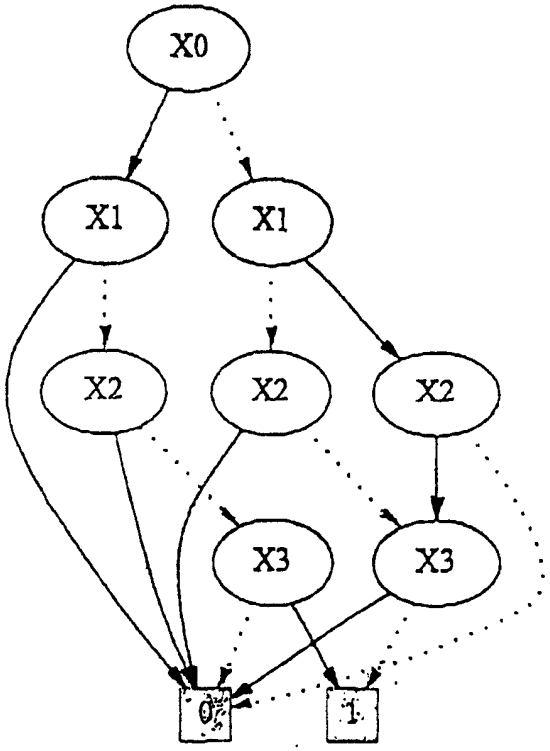


图 8

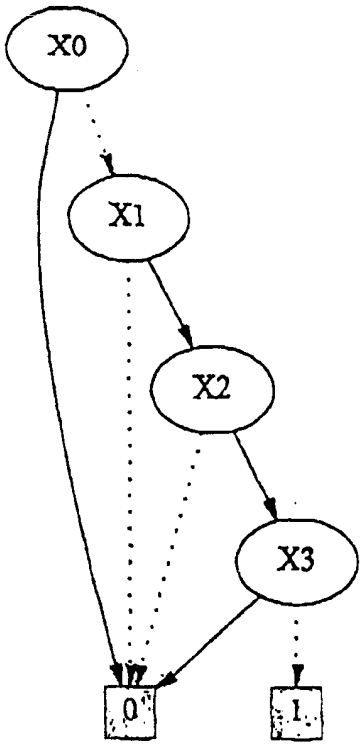


图 9

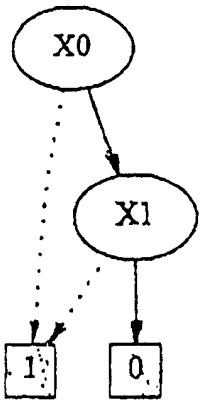


图 10