第5章

逻辑综合

逻辑综合(Logic Synthesis)是从寄存器传输级到晶体管级过渡过程中发生的过程。它是一个高度自动化的过程,是连接了高层综合与物理设计自动化之间的桥梁。给定寄存器传输级的数字设计,逻辑综合将其转换为门级或晶体管级的实现。这个过程会探索实现逻辑功能的不同方式,以满足某些设计约束的最优条件。门布局的物理位置和互连在物理设计时进一步确定。

5.1 概　　述

逻辑综合的主要数学基础是逻辑和代数的交集。乔治·布尔(George Boole)于1847年创立的"逻辑代数"(Boolean Algebra,又称为布尔代数)是逻辑综合的核心。在本书中,将专注于两元素布尔代数[1]。将布尔代数与电路设计联系起来的最具影响力的工作之一是克劳德·E·香农(Claude E. Shannon)的硕士论文《对继电器和开关电路中的符号分析》("A Symbolic Analysis of Relay and Switching Circuits"),该论文于1937年完成于麻省理工学院。香农展示了开关电路的设计和分析可以通过布尔代数形式化,并且开关电路可以用来解决布尔代数问题。基于数字(与模拟相对)和二值(与多值相对)原理的现代电子系统在很大程度上可以归功于香农的工作。布尔公式在两级与或表达式(Sum-Of-Products,SOP)形式下的最小化理论由威拉德·V·奎因(Willard V. Quine)在20世纪50年代建立。20世纪70年代,SOP公式的最小化在IC设计中得到了广泛应用,当时可编程逻辑阵列(PLAs)是一种用于控制逻辑实现的流行设计风格。这是逻辑设计最小化的最早阶段。当多级逻辑实现在20世纪80年代变得可行时,最小化理论和实践被拓展到了多级场景。

除了两级和多级逻辑最小化之外,20世纪80年代,逻辑综合的重要算法发展还包括同步时序电路的重新定时、工艺映射算法、简化有序二叉决策图,以及使用特征函数的符号顺序等价性检查等。这个时期的主要逻辑综合工具包括加州大学伯克利分校开发的ESPRESSO[11]和后来的MIS[10]。它们很快成为商业逻辑综合工具的核心引擎。

20世纪90年代,逻辑综合的主题因各种集成电路设计问题而变得多样化:功耗、互连延迟、可测试性、新的实现风格如现场可编程门阵列(FPGA)等。这个时期的重要算法突破包括时序电路综合中的重新定时和再综合、无关项计算、时

序分析、布尔推理技术等。这个时期开发的主要学术软件包括 SIS[12]，是 MIS 的后继者。21 世纪，逻辑综合的发展方向受到设计挑战的驱动，如可扩展性、可验证性、逻辑综合与物理设计之间的设计闭合问题、制造工艺变化等。重要发展包括有效的可满足性求解程序、可扩展的逻辑综合和验证算法、统计静态时序分析、统计优化技术等。这个时期开发的主要学术软件包括 MVSIS 和 ABC 包[9]，首次发布于 2005 年。逻辑综合的进步反过来促进了 EDA 公司的兴起和 EDA 产业的增长。逻辑综合在商业应用中的首次应用之一是将网表重新映射到不同的标准单元库（1986 年成立的 EDA 公司 Synopsys 开发的第一个产品 remapper）。这使集成电路设计师能够将设计从一个库迁移到另一个库。逻辑优化用于优化门级网表并将其映射到目标库中。

逻辑综合的范围可以这样定义。一个 IC 可能由数字和模拟组件组成，逻辑综合关注的是数字部分。对于具有时序行为的数字系统，其状态转换可以根据是否存在同步时钟信号以同步或异步方式实现（注意，即使是组合电路也可以被视为单状态时序系统）。大多数逻辑综合算法专注于同步实现，少部分关注异步实现。数字系统通常可以分为两部分：数据路径和控制逻辑。前者涉及数据计算和存储，通常由算术逻辑单元、总线、寄存器/寄存器文件等组成；后者负责控制这些数据处理单元。与控制逻辑不同，数据路径电路通常由规则结构组成。为了确保设计约束得到满足，尤其是在高性能应用中，IC 设计师通常手动进行全定制设计。因此，数据路径设计涉及较少的逻辑综合工作。相比之下，控制逻辑通常使用逻辑综合来设计。逻辑综合的优势在于其强大的逻辑最小化能力，能够显著简化控制逻辑。因此，逻辑综合特别适用于以控制为主的应用，如协议处理，但不适用于以算术运算为主的应用，如信号处理。

除了电路组件相关的设计问题外，市场导向的决策也影响产品实现中选择的设计风格。设计自动化和逻辑综合工作量在很大程度上依赖于这些决策。在全定制设计中，逻辑综合用途有限，主要用于综合性能不关键的控制器。对于基于标准单元和 FPGA 设计，设计大部分可能通过逻辑综合处理。因此，逻辑综合广泛应用于专用集成电路（ASIC）和基于 FPGA 设计中。

5.2　布尔表示和推理的数据结构

本章要处理的基本数学对象是布尔函数。如何紧凑地表示布尔函数（逻辑最小化的主题）以及如何高效地解决布尔约束（布尔推理的主题）是密切相关的问题，在逻辑综合中起着核心作用。有几种用于布尔函数表示和操作的数据结构。对于布尔表示，我们介绍一些最常用的结构，特别是积之和形式（SOP）、和之积形式（POS）、二叉决策图（BDDs）、与-非图（AIGs）以及布尔网络等。对于布尔推理，我们讨论如何利用 BDD、SAT 和 AIG 包作为布尔函数操作和自动推理布尔函数性质的核心引擎。数据结构的效率主要取决于其在表示布尔函数时的简洁性以及支持布尔操作的能力。每种数据结构都有其自身的优缺点；没有一种数据结构能够在所有应用中表现出色。因此，在逻辑综合中，不同数据类型之间的转换是必要的，所以需要使用各种电路转换和验证技术。

5.2.1 无量化和量化布尔公式

我们引入无量化布尔公式用于布尔函数表示，量化布尔公式(QBFs)用于布尔推理。一个布尔变量是仅取二值集合 $B=\{\text{false},\text{true}\}$ 的变量，其值由真值赋值确定；一个文字是一个布尔变量或其补变量。在 n 维布尔空间 B^n 中，一个原子元素(或顶点)$a\in B^n$ 被称为一个最小项，它对应包含 n 个布尔变量的向量的一种真值赋值。

一个完全指定的 n 元布尔函数 $f:B^n\to B$ 将 n 个输入变量的每一种可能真值赋值映射为 true 或 false。我们用符号“－”“X”或“2”表示无关值。将 B 扩展为 $B_+=B\cup\{-\}$，并定义一个不完全指定的布尔函数 $f:B^n\to B_+$，该函数将 n 个输入变量的每一种可能的真值赋值映射为 true、false 或 don't care(无关)。对于某些 $a\in B^n$，如果 $f(a)=-$，则表示在真值赋值 a 下，函数 f 的值无关紧要。换句话说，a 是 f 的一个无关条件。除非另有说明，我们假定布尔函数是完全指定的。

由一组布尔函数定义的映射可以用一个函数向量或一个多输出函数来描述。如果 f 是完全指定的，它将多个布尔函数组合成一个映射 $f:B^n\to B^m$(其中 $m>1$)；如果 f 是不完全指定的，则映射为 $f:B^n\to B_+^m$。对于一个完全指定的布尔函数 f，我们定义其正集合为 $f_{\text{on}}=\{a\in B^n\mid f(a)=1\}$，其负集合为 $f_{\text{off}}=\{a\in B^n\mid f(a)=0\}$。对于一个不完全指定的布尔函数 f，除了正集合和负集合外，还定义了无关集合为 $f_{\text{dc}}=\{a\in B^n\mid f(a)=-\}$。

例 5.1 由变量向量(x_1,x_2,x_3)展开的布尔三维空间可以看作一个组合立方体，如图 5.1 所示，其中标记的顶点表示最小项，两个最小项通过边相连，当且仅当它们的汉明距离为 1(即它们的二进制编码在某一位上不同)。某个函数 f 的正集 $f_{\text{on}}=\{000,011,100,101,110\}$，负集 $f_{\text{off}}=\{001,111\}$，以及无关集 $f_{\text{dc}}=\{010\}$嵌入在组合立方体中。

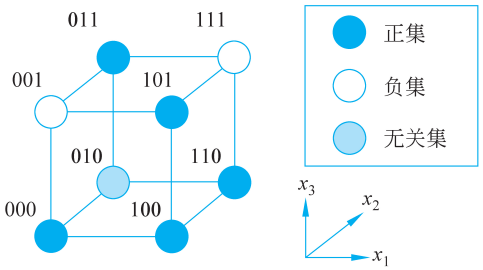


图 5.1 布尔三维空间和一个三元布尔函数

一个完全指定的布尔函数 f 是一个重言式，当且仅当 $f\equiv 1$ 或 $f\Leftrightarrow 1$ 即它的正集等于全集，也就是整个布尔空间，在这种情况下，函数 f 的输出在输入变量的每一个真值赋值下都等于 1。任何布尔函数都可以用一个布尔公式表示。表 5.1 展示了布尔公式的构建元素(不包括最后两个符号 $\exists$ 和 $\forall$)。符号 $\neg$、$\wedge$、$\vee$、$\Rightarrow$、$\Leftrightarrow$是布尔连接词。一个布尔公式 φ 可以通过以下递归规则构建。

$$\varphi ::= 0\mid 1\mid A\mid \neg\varphi_1\mid \varphi_1\wedge\varphi_2\mid \varphi_1\vee\varphi_2\mid \varphi_1\Rightarrow\varphi_2\mid \varphi_1\Leftrightarrow\varphi_2 \tag{5.1}$$

其中，符号“::=”表示“可以是”，符号“|”表示“或”。也就是说，一个布尔公式 φ 可以是常量 0、常量 1、来自变量集合 A 的原子布尔变量、$\neg\varphi_1$、$\varphi_1\wedge\varphi_2$、$\varphi_1\vee\varphi_2$、$\varphi_1\Rightarrow\varphi_2$ 或 $\varphi_1\Leftrightarrow\varphi_2$，这些都可以从布尔公式 φ_1 和 φ_2 构建。为了省略括号并提高可读性，假设布尔连接词$\Leftrightarrow$、$\Rightarrow$、$\vee$、$\wedge$、$\neg$的优先级按降序排列。此外，我们经常省略公式中布尔连接词 $\wedge$ 的显式表示。

表 5.1　符号表示及意义

符　　号	符 号 名 称	中 文 意 义
(	左括号	用于标点
)	右括号	用于标点
¬，’	补集符号	逻辑“非”
∧，·	合取符号	逻辑“与”
∨，＋	析取符号	逻辑“或”
⇒	蕴含符号	逻辑“如果……，那么……”
⇔，==	双条件符号	逻辑“……当且仅当……”
∃	存在量词	“存在……”
∀	全称量词	“对于所有……”

例 5.2　布尔公式$((x_1 \vee (\neg x_2)) \vee ((\neg x_1) \wedge x_3)) \wedge (x_1 \wedge (\neg x_2))$可以简化为$((x_1 \vee \neg x_2) \vee \neg x_1 x_3)(x_1 \neg x_2)$。使用析取与合取的结合律，可以进一步将公式简化为$(x_1 \vee \neg x_2 \vee \neg x_1 x_3) x_1 \neg x_2$，但无法再追踪用于推导此公式的唯一规则序列。

一组布尔运算符足以生成任何布尔函数，可以称其为功能完备。需要注意的是，并非所有的布尔连接词构成一个功能完备的运算符集合都是必要的。例如，集合$\{\neg, \wedge\}$和$\{\neg, \Rightarrow\}$是功能完备的，而$\{\wedge, \Rightarrow\}$则不是。布尔函数可以看作某些布尔公式的语义（即它们的逻辑意义）。不同的布尔公式（从语法角度看可能有差异）可以表达相同的布尔函数（从逻辑意义上看是等价的）。这种语法与语义的多样性和灵活性，使得逻辑综合成为一种需要技巧和创造力的过程。布尔函数上的布尔运算可以用集合运算来定义，如集合的并集$\cup$、交集$\cap$和补集。布尔函数$h = f \wedge g$的“正集合”为$h_{on} = f_{on} \cap g_{on}$，而“负集合”为$h_{off} = f_{off} \cap g_{off}$。布尔函数$h = \neg f$（也记作$h = f'$）的正集为$h_{on} = f_{off}$，关闭集为$h_{off} = f_{on}$。函数$h$的“无关集合”可以通过开启集和关闭集的并集来推导，且“无关集合”等于全集。

量化布尔公式（Quantified Boolean Formulas，QBFs）是对无量化布尔公式的推广，增加了全称量化符号∀和存在量化符号∃。在书写 QBF 时，假设量化符号的优先级高于布尔连接词。在一个 QBF 中，被量化的变量称为绑定变量，而未被量化的变量称为自由变量。

例 5.3　考虑量化布尔公式（QBF）：$\forall x_1, \exists x_2 . f(x_1, x_2, x_3)$，其中$f$是一个布尔公式。这可以读作：对于每个$x_1$的真值分配，存在某个$x_2$的真值分配，使得$f(x_1, x_2, x_3)$为真。在这种情况下，$x_1$和$x_2$是绑定变量，而$x_3$是自由变量。

任何 QBF 都可以通过量化符号消除的方法（如公式展开）重写为无量化符号的布尔公式，例如，$\forall x . f(x, y) = f(0, y) \wedge f(1, y)$以及$\exists x . f(x, y) = f(0, y) \vee f(1, y)$，其中，$f$是一个布尔公式。因此，对于任何 QBF ϕ，都存在一个等价的无量化布尔公式，该公式仅涉及ϕ的自由变量。对于一个大小为n且具有k个绑定变量的 QBF，通过公式展开得到的无量化布尔公式的大小可以达到$O(2^n \cdot k)$。QBF 的表达能力与无量化布尔公式相同，但其形式可以更为紧凑。

例 5.4　量化布尔公式$\forall x_1, \exists x_2 . f(x_1, x_2, x_3)$可以重写为$\forall x_1 . (f(x_1, 0, x_3) \vee f(x_1, 1, x_3)) = (\exists x_2 . f(0, x_2, x_3)) \wedge (\exists x_2 . f(1, x_2, x_3)) = (f(0, 0, x_3) \vee f(0, 1, x_3)) \wedge$

$(f(1,0,x_3) \vee f(1,1,x_3))$。

注意，$\forall x_1, \exists x_2.f(x_1,x_2,x_3)$与$\exists x_2, \forall x_1.f(x_1,x_2,x_3)$不同，并且实际上比后者更弱。也就是说，$[(\exists x_2, \forall x_1.f(x_1,x_2,x_3)) \Rightarrow (\forall x_1, \exists x_2.f(x_1,x_2,x_3))]$。相反，$\forall x_1, \forall x_2.f(x_1,x_2,x_3)$等价于$\forall x_2, \forall x_1.f(x_1,x_2,x_3)$，类似地，$\exists x_1, \exists x_2.f(x_1,x_2,x_3)$等价于$\exists x_2, \exists x_1.f(x_1,x_2,x_3)$。

此外，可以验证，全称量化符号$\forall$与合取($\wedge$)是可交换的，而存在量化符号$\exists$与析取($\vee$)是可交换的。也就是说，对于任意的 QBF φ_2 和 φ_2 有 $\forall x.(\varphi_1 \wedge \varphi_2) = \forall x.\varphi_1 \wedge \forall x.\varphi_2$，而$\exists x.(\varphi_1 \vee \varphi_2) = \exists x.\varphi_1 \vee \exists x.\varphi_2$。然而，一般情况下，全称量化符号$\forall$并不与析取($\vee$)可交换，而存在量化符号$\exists$也不与合取($\wedge$)可交换。也就是说，一般情况下，$\forall x.(\varphi_1 \vee \varphi_2) \neq \forall x.\varphi_1 \vee \forall x.\varphi_2$，并且$\exists x.(\varphi_1 \wedge \varphi_2) \neq \exists x.\varphi_1 \wedge \exists x.\varphi_2$。另一方面，对于任意的 QBF φ，有

$$\neg \forall x.\varphi = \exists x.\neg\varphi \tag{5.2}$$

并且

$$\neg \exists x.\varphi = \forall x.\neg\varphi \tag{5.3}$$

由于$\forall$和$\exists$可以通过取反相互转换，因此仅使用其中一个量化符号就足以表示 QBF。关于 QBF 的一个重要事实是：在重命名约束变量的情况下，它们是等价的。例如，$\forall x.f(x,y) = \forall z.f(z,y)$和$\exists x.f(x,y) = \exists z.f(z,y)$。如果想以不同的方式重写一个 QBF，通常需要重命名约束变量。能够识别量化符号的作用域对于这样的重命名至关重要。

例 5.5 在 QBF 中，$Q_1x, Q_2y.(f_1(x,y,z) \vee f_2(y,z) \wedge Q_3x.f_3(x,y,z))$，其中，$Q_i \in \{\forall, \exists\}$，量化符号 Q_1 仅作用于 f_1 的变量 x，量化符号 Q_2 仅作用于所有函数的变量 y，量化符号 Q_3 仅作用于 f_3 的变量 x。该 QBF 可以重命名为 $Q_1a, Q_2b.(f_1(a,b,z) \vee \neg f_2(b,z) \wedge Q_3x.f_3(x,b,z))$。

在研究 QBF 时，引入一种统一的表示形式，即所谓的前束范式，是很方便的。在这种形式中，QBF 的所有量化符号被移到公式的左侧，而右侧是一个无量化的布尔公式。即 $Q_1x_1, Q_2x_2, \cdots, Q_nx_nf(x_1,x_2,\cdots,x_n)$，其中，$Q_i \in \{\forall, \exists\}$，$f$ 是一个无量化的布尔公式。任何 QBF 都可以转换为前束范式的等价公式。前束范式特别适合研究计算复杂性。在前束范式的 QBF 中，存在量词和全称量词之间交替次数直接反映了解决问题的难度。

在求解一个 QBF 公式 φ 时，假设 φ 的所有变量都是量化的(即 φ 中没有自由变量)。例如，量化布尔公式 QBF $\forall x_1, \forall x_2, \exists x_3, \forall x_4, \exists x_5.f(x_1,x_2,\cdots,x_5)$中存在三个量化交替。前束范式中 QBF 的量化交替次数越多，其求解的计算复杂性越高。这种难度的层次引出了多项式层级，即复杂性理论中的复杂性类层次[6]。求解 QBF 的问题被称为量化可满足性问题(Quantified SATisfiability，QSAT)。特别地，对于具有 i 次量化交替的前束范式 QBF，其问题被称为 QSATi。整个多项式层级包含于 PSPACE 复杂性类中；而不预先限定交替次数 i 的问题 QSAT 是 PSPACE 中最难的问题之一，即 PSPACE 完全问题。一个特别有趣的特例是 QSAT00，其中所有变量都以存在量化的方式出现。这被称为布尔可满足性问题(Boolean Satisfiability，SAT)，其是一个 NP 完全问题[4]。求解 QBF 比求解布尔公式的可满足性要困难得多。

5.2.2 布尔函数操作

除了布尔的与、或、非运算外，余因子是一种基本的布尔操作。对于一个函数 $f(x_1,$

$x_2,\cdots,x_i,\cdots,x_n)$，相对于变量 x_i 的正余因子和负余因子分别为 $f(x_1,x_2,\cdots,1,\cdots,x_n)$，记作 f_{x_i} 或 $f|_{x_i=1}$；$f(x_1,x_2,\cdots,0,\cdots,x_n)$，记作 $f_{\neg x_i}$ 或 $f|_{x_i=0}$。此外，还可以相对于一个立方体(cube)对布尔函数进行余因子化。所谓立方体，是一组文字的合取式。通过逐步对函数应用立方体中的每个文字，可以得到相对于该立方体的余因子。

例 5.6　对布尔函数 $f=x_1x_2\neg x_3\vee x_4\neg x_5x_6$ 相对于立方体 $c=x_1x_2\neg x_5$ 进行余因子化，得到函数 $f_c=\neg x_3\vee x_4x_6$。

全称量化和存在量化可以用余因子表示，其中，

$$\forall x_i: f=f_{x_i}\wedge f_{\neg x_i} \tag{5.4}$$

$$\exists x_i: f=f_{x_i}\vee f_{\neg x_i} \tag{5.5}$$

此外，相对于变量 x_i 的布尔差分 $\dfrac{\partial f}{\partial x_i}$ 定义为

$$\frac{\partial f}{\partial x_i}=\neg(f_{x_i}\equiv f_{\neg x_i})=f_{x_i}\oplus f_{\neg x_i} \tag{5.6}$$

其中，$\oplus$ 表示异或(XOR)运算。通过布尔差分运算，可以判断布尔函数是否在功能上依赖于某个变量。如果 $\dfrac{\partial f}{\partial x_i}$ 等于常量 0，那么 f 值不依赖于 x_i，也就是说，x_i 是 f 的冗余变量。如果 x_i 不是冗余变量，称 x_i 是 f 的功能支持变量。根据香农展开，每个布尔函数 f 都可以相对于某个变量 x_i 表示为

$$f=x_if_{x_i}\vee\neg x_if_{\neg x_i} \tag{5.7}$$

注意，变量 x_i 不一定是 f 的功能支持变量。

5.2.3　布尔函数表示

下面讨论表示布尔函数的不同方法。

1. 真值表

布尔函数的映射可以通过真值表进行穷举枚举，其中每一个真值赋值都有一个对应的函数值列出。

例 5.7　图 5.2 显示了多数函数 $f(x_1,x_2,x_3)$ 的真值表，该函数在且仅在变量 x_1、x_2、x_3 中至少两个变量为真时取值为真，真值表是布尔函数的规范表示。也就是说，两个布尔函数当且仅当它们有相同的真值表时是等价的。

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

图 5.2　多数函数的真值表

规范性是逻辑综合和验证的许多应用中可能有用的重要属性。在实际实现中，真值表在表示具有少量输入变量(对于现代计算机来说，通常不超过 5 或 6 个变量)的函数时非常有效，因为现代计算机的字长为 32 位或 64 位。通过将真值表存储为计算机字，两个小函数之间的基本布尔操作可以通过它们的真值表的并行按位操作在常数时间内完成。然而，对于具有许多输入变量的函数来说，使用真值表表示是不切实际的。

2. SOP

积之和(Sum-of-Products，SOP)或析取范式(Disjunctive Normal Form，在计算机科学中称为 DNF)是布尔公式的一种特殊形式，由文字的合取(乘积项或立方体)的析取(和)组成。这是一种平坦的结构，对应于两级电路表示(第一级是与门，第二级是或门)。在两级逻辑最小化中，SOP 表示的布尔函数的乘积项集合(即立方体)称为布尔函数的覆盖。一个布尔函数可能有许多不同的覆盖，而一个覆盖唯一地确定一个布尔函数。

例 5.8 表达式 $f=ab\neg c+a\neg bc+\neg abc+\neg a\neg b\neg c$ 是 SOP 形式。立方体集合$\{ab\neg c, a\neg bc, \neg abc, \neg a\neg b\neg c\}$构成了函数 f 的一个覆盖。

在我们的讨论中，通常不区分一个覆盖和它所表示的函数。每个布尔公式都可以重写为 SOP 表示形式。与真值表表示不同，SOP 表示形式不是规范的。事实上，如何以最简洁的 SOP 形式表达一个布尔函数是难以处理的(实际上是 NP 完全问题)，这被称为两级逻辑最小化。

以 SOP 作为布尔表示的基础，我们研究其在布尔操作中的用途。考虑两个立方体的合取。由于立方体被定义为文字集合，计算两个立方体 c 和 d 的合取(记为 $q=c\cap d$)可以通过取 c 和 d 的文字集合的并集来实现，因此其计算时间是与文字数量线性相关的。然而，如果通过这种方式计算的 $q=c\cap d$ 包含某个文字 l 及其补文字 $\neg l$，那么这个交集是空的。同样地，两个覆盖的合取可以通过取覆盖中每对立方体的交集来获得。因此，两个 SOP 公式的与操作(AND 操作)的时间复杂度是二次的。另一方面，或操作(OR 操作)的时间复杂度是常数，因为两个 SOP 公式的析取可以直接以 SOP 形式表示。补操作在最坏情况下的时间复杂度是指数级的。

例 5.9 对函数 $f=x_1\cdot y_1+x_2\cdot y_2+\cdots+x_n\cdot y_n$ 取补，将在 SOP 表示形式中产生 2^n 个乘积项。

除了上述基本的布尔运算外，可满足性检查和重言式检查在布尔推理中也起着核心作用。检查一个 SOP 公式是否可满足的时间复杂度是常数时间，因为任何(无冗余的)SOP 公式，只要不恒等于 0，就一定是可满足的。相比之下，检查一个 SOP 公式是否是重言式实际上是一个 coNP 完全问题。与即将介绍的其他数据结构相比，SOP 通常并不被用作布尔推理引擎中的底层表示，而主要用于两级和多级逻辑最小化中。为了实现两级逻辑函数的最小化，开发高效的算法以在 SOP 表示或覆盖上执行布尔运算是非常必要的。一个用于执行各种布尔运算(如合取、析取和取补)的工具包作为 ESPRESSO 程序的一部分提供[10]。

3. POS

和之积(Product-of-Sums，POS)，或称为合取范式(Conjunctive Normal Form，CNF)，在计算机科学中是一种特殊形式的布尔公式。它由文字(子句)的析取(和)组成的合取(积)构成。这是一种平坦结构，对应两级电路表示(第一级是或门，第二级是与门)。

例 5.10 公式$(a+b+\neg c)(a+\neg b+c)(\neg a+b+c)(\neg a+\neg b+\neg c)$是 POS 形式。

每个布尔公式都有一个等价的 POS 形式。尽管 POS 看起来只是 SOP 的对偶形式，但它在电路设计中不像 SOP 那样常用，这部分是由于 CMOS 电路的特性，在 CMOS 中，NMOS 比 PMOS 更优。尽管如此，POS 在布尔推理中被广泛使用。求解 CNF 公式的可满足性(SAT)是计算机科学中最重要的问题之一。实际上，每个 NP 完全问题都可以在多项式时间内转换为 SAT 问题。以 POS 作为底层数据结构，我们研究其在布尔函数操作中的作用。对于与操作(AND)，由于两个 POS 公式的合取在 POS 中可以直接表示，其时间复杂度是常数时间。对于或操作(OR)，由于在最坏情况下两个 POS 公式的析取需要通过分配律转换为 POS 形式，其时间复杂度是二次时间。

例 5.11　给定 POS 公式 $\varphi_1=(a)\cdot(b)$ 和 $\varphi_2=(c)\cdot(d)$，它们的析取 $(\varphi_1+\varphi_2)$ 等于 $(a+c)\cdot(a+d)\cdot(b+c)\cdot(b+d)$。

另外，取补操作的时间复杂度是指数级的，因为在最坏情况下，一个 POS 公式可能需要通过德摩根定律(De Morgan's Law)和分配律逐步取补。对于 POS 公式的可满足性检查和恒真性检查，前者是 NP 完全问题，而后者的时间复杂度是常数时间，因为除了常数 1 以外的任何(无冗余的)POS 公式都不可能是恒真的。POS 表示通常被用作布尔推理引擎(称为 SAT 求解器)中的底层表示形式。

4. BDD

二叉决策图(Binary Decision Diagrams，BDDs)最初由 Lee 提出，并由 Akers 进一步发展。在其原始形式中，BDD 并不是布尔函数表示的规范化形式。为了规范化表示，Bryant 对 BDD 的变量顺序引入了限制，并提出了若干化简规则，从而得到了广为人知的约简有序二叉决策图(Reduced Ordered BDDs，ROBDDs)。在各种决策图类型中，ROBDDs 是最广泛使用的。考虑使用一个 n 层二叉树来表示任意 n 输入布尔函数 $f(x_1,x_2,\cdots,x_n)$。这种二叉树称为 BDD，包含两种类型的节点：终端节点，或称为叶节点，记为 v，具有一个属性值 $\text{value}(v)\in\{0,1\}$；非终端节点。$v$ 具有以下属性：一个参数，层级索引 $\text{index}(v)\in\{1,2,\cdots,n\}$，表示变量索引；两个子节点，0 子节点，记为 $\text{else}(v)\in V$，1 子节点，记为 $\text{then}(v)\in V$。如果 $\text{index}(v)=i$，则 x_i 被称为节点 v 的决策变量。BDD 中的每个节点 v 对应于通过如下递归定义的布尔函数 $f[v]$。

(1) 对于一个终端节点 v：

① 如果 $\text{value}(v)=1$，那么 $f[v]=1$。

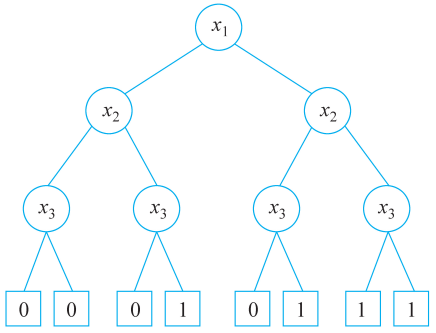
② 如果 $\text{value}(v)=0$，那么 $f[v]=0$。

(2) 对于一个非终端节点 v，其 $\text{index}(v)=i$：

$$f[v](x_1,x_2,\cdots,x_n)=\neg x_i\cdot f[\text{else}(v)](x_1,x_2,\cdots,x_n)+x_i\cdot f[\text{then}(v)](x_1,x_2,\cdots,x_n)$$

回忆香农展开，布尔函数 f 可以表示为 $f=x_i f_{x_i}+\neg x_i f_{\neg x_i}$。假设一个 BDD 节点表示某个布尔函数 f，并由变量 x_i 控制，那么它的 0 子节点和 1 子节点分别表示函数 $f_{\neg x_i}$ 和 f_{x_i}。因此，BDD 实质上表示了一个递归的香农展开。对于一个完整的二叉树，很容易看出，我们总能找到某种值分配方式，使得 BDD 的叶节点实现任意 n 输入的布尔函数 $f(x_1,x_2,\cdots,x_n)$。这是因为每个变量 $x_1,x_2,\cdots,x_n$ 的具体赋值会激活从根节点到唯一叶节点的一条路径，并且该叶节点具有正确的函数值。需要注意的是，BDD 表示了一个函数的正集和负集作为不相交的覆盖，其中覆盖中的每个立方体对应从根节点到某个终端节点的路径。

例 5.12 多数函数的二叉树表示如图 5.3 所示，其中，圆形（方形）表示一个非终端节点（终端节点），而虚线（实线）边表示从父节点指向的 0 子节点（1 子节点）。



x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

图 5.3 多数函数的二叉树表示

定义 5.1 当从 BDD 的根节点到终端节点的每条路径都遵循相同的变量顺序时，该 BDD 被称为有序的（即 OBDD）。

定义 5.2 如果存在一个从 D_1 的节点到 D_2 的节点的一一映射函数 σ，使得对于任意节点 v，如果 $\sigma(v)=w$，在以下条件成立，则 D_1 和 D_2 两个 OBDD 是同构的，v 和 w 要么都是终端节点，且 value(v)=value(w)；要么都是非终端节点，且 index(v)=index(w)，σ(else(v))=else(w)，σ(then(v))=then(w)。

由于 OBDD 仅包含一个根节点，并且任何非终端节点的子节点都是可区分的，因此 D_1 和 D_2 之间的同构映射 σ 是受约束且易于检查的。例如，D_1 的根节点必须映射到 D_2 的根节点，D_1 根节点的 0 子节点必须映射到 D_2 根节点的 0 子节点，以此类推，直到终端节点。因此，测试两个 OBDD 是否同构是一个简单的线性时间检查过程。

定义 5.3 当一个 OBDD 不包含满足 else(v)=then(v)的节点 v，也不包含两个不同的节点 v 和 w 且以它们为根的子图是同构时，该 OBDD 被称为化简的。一个化简的 OBDD（ROBDD）可以通过以下三条化简规则从一个 OBDD 构造得到。

（1）具有相同值属性的两个终端节点将被合并。

（2）具有相同决策变量、相同 0 子节点（即 else(u)=else(v)）以及相同 1 子节点（then(u)=then(v)）的两个非终端节点 u 和 v 将被合并。

（3）满足 else(v)=then(v)的非终端节点 v 将被移除，其相关边将重新定向到其子节点。

对 OBDD 自底向上迭代应用上述化简步骤，直到无法进行进一步修改时，将得到其唯一对应 ROBDD。规则确保 ROBDD 中的任意两个节点在结构上（功能上）都不是同构的，并且在给定变量排序下，派生的 ROBDD 具有最少的节点数量。这表明，ROBDD 中的任意两个节点都不能表示相同的布尔函数，因此两个表示相同布尔函数的 ROBDD 必须是同构的。ROBDD 是布尔函数规范表示，在给定变量排序下，每个函数都有唯一的 ROBDD。

定理 5.1 ROBDD 的规范性[14]。对于任意布尔函数 f，存在唯一的（同构意义下）表示 f 的 ROBDD，并且任何其他表示 f 的 OBDD 都包含更多的节点。

例 5.13 图 5.4（从图 5.4(a)到图 5.4(c)）展示了从多数函数的二叉树到 ROBDD 推导过程。

例 5.14　考虑图 5.5(a)中的 OBDD。根据第一条化简规则，可以合并所有值为 0 的终端节点以及所有值为 1 的终端节点。两个以控制变量 x_3 为根的节点是相同的，即 x_3。根据第二条化简规则，可以删除其中一个相同的节点，并使指向被删除节点的节点(即 0 子节点或 1 子节点指向被删除节点的那些节点)改为指向另一个节点。这不会改变 OBDD 所表示的布尔函数。化简后的 OBDD 如图 5.5(b)所示。在图 5.5(b)中，有一个以控制变量 x_2 为根的节点，其 0 子节点和 1 子节点都指向同一个节点。这个节点是冗余的，因为以该节点为根的布尔函数与其子节点所表示的函数相同。

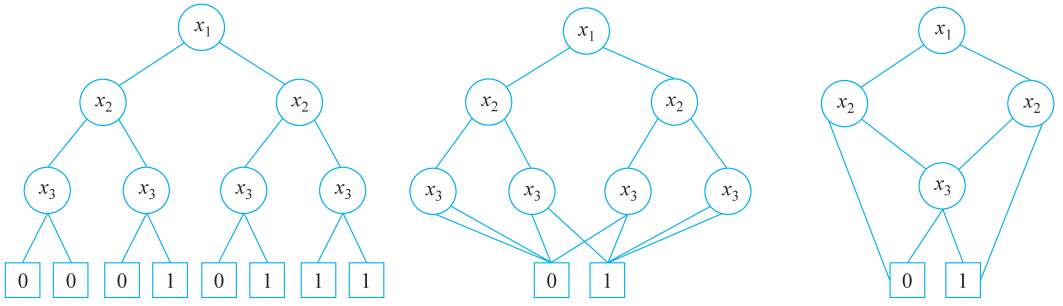


图 5.4　二叉树到 ROBDD

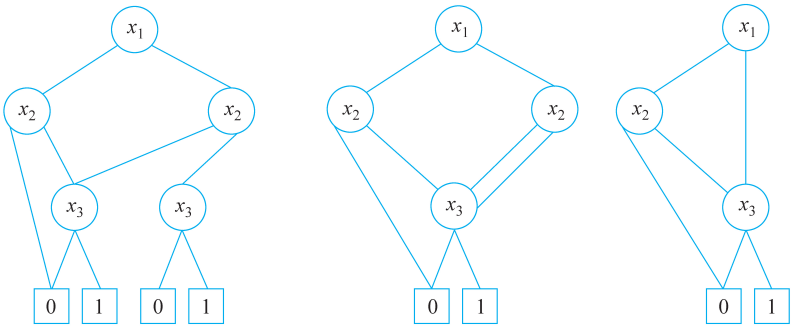


图 5.5　化简 $f=x_2 \cdot x_3 + \neg x_2 \cdot x_3 = x_3$

例 5.15　图 5.6 展示了一个使用标记技术对 ROBDD 进行化简的示例[14]。首先将图 5.6(a)中的 0 和 1 终端节点分别赋予标记 a 和 b。接下来，将控制变量为 x_3 的右节点赋予标记 c。当遇到另一个控制变量为 x_3 的节点时，发现满足第二条化简规则，因此也为该节点赋予标记 c。继续向上，为控制变量为 x_2 的右节点赋予标记 c，因为该节点满足第三条化简规则(该节点的 0 子节点和 1 子节点具有相同的标记)。控制变量为 x_2 的左节点被赋予标记 d，根节点被赋予标记 e。请注意，节点的标记方式使得每个标记对应一个唯一的(子)ROBDD。对节点进行排序并删除冗余节点会产生图 5.6(b)中的 ROBDD。

例 5.16　为了说明 ROBDD 如何将函数的负集和正集表示为不相交的覆盖集，请参考图 5.7 示例。图 5.7(a)中 ROBDD 表示函数 $f=x_1 \wedge x_2$。该 ROBDD 中恰好有两条路径通向 0 终端节点。如果 $x_1=0$，则由 ROBDD 表示的函数的值为 0，因为索引为 x_1 的节点的 0 子节点是 0 终端节点。如果 $x_1=1$ 且 $x_2=0$，函数值也为 0。因此偏移量可以表示为 $\{\neg x_1, x_1 \wedge \neg x_2\}$。覆盖集中包含的两个立方体是互不相交的。如果 x_1 和 x_2 都为 1，函数