二元判决图的实现及改进方法*

李翔宇** 陈光禹
（电子科技大学 CAT 室 成都 610054）

【摘要】 构造布尔函数的二元判决图(BDD)的核心是 ite 算符，文中详细介绍了 ite 算符表示布尔代数的基本运算，以及 ite 算符在构造 BDD 中的作用和实现方法；讨论了运用哈希表、反向属性边等策略提高软件包性能的方法。试验结果表明，该 BDD 软件包性能优于国外同类软件。

关 键 词 二元判决图； 运算符； 哈希表； 反向属性边

中图分类号 TP31

二元判决图(Binary Decision Diagram——BDD)是一种对布尔函数进行运算的有效方法[1, 2]，在电路的故障检测、逻辑验证、逻辑综合等诸多方面得到了广泛的应用。构造电路的 BDD 是所有应用的基础。我们在 PC 平台上用 C++语言独立实现了一个通用 BDD 包，该包综合引用了国外的一些先进技术[1, 3, 4]。本文讨论 BDD 运算包的实现基础——ite 算符，介绍了 BDD 包的实现流程，以及改善其性能的一些方法，并给出试验结果。

1 ite 算符及算法

布尔代数的基本运算包括与、或、非、异或、异或非等，为了用一种统一的形式来表示这些基本运算，引入 ite（IF-THEN-ELSE）运算符，这是一个三元逻辑运算符，定义为

$$\mathrm{ite}(f,g,h)=f\,g+\overline{f}\,h \tag{1}$$

所有的一元二元逻辑运算都可以用 ite 算符来实现，例如:

$\overline{x}=\mathrm{ite}(x,0,1)$; $x_1\cdot x_2=\mathrm{ite}(x_1,x_2,0)$; $x_1+x_2=\mathrm{ite}(x_1,1,x_2)$;

$x_1\oplus x_2=\mathrm{ite}(x_1,\overline{x_2},x_2)$; $x_1\overline{\oplus}x_2=\mathrm{ite}(x_1,x_2,\overline{x_2})$;

$\overline{x_1\cdot x_2}=\mathrm{ite}(x_1,\overline{x_2},1)$; $\overline{x_1+x_2}=\mathrm{ite}(x_1,0,\overline{x_2})$ 。

以这种统一的形式来实现二元逻辑运算，可以识别运算中更多的等效形式。

给定函数 f，g，h 的 BDD 表达 F，G，H，计算 ite (F，G，H)时，不能使用 ite 的定义式，因为它只是把三元逻辑运算分解为几个二元逻辑运算，必须采用递归方法才能利用 ite 算符的优越性。

在函数 f: $\{x_1,x_2,\cdots,x_n\}\to\{0,1\}$用二元判决图 F 表示时，香农展开式为

$$F=wF_w+\overline{w}F_{\overline{w}} \tag{2}$$

式中 $w\in[1,n]$，F_w 和 $F_{\overline{w}}$ 分别对应 F 在所有序为 w 的变量取 1 和 0 后产生的 BDD。根据式(2)，可得递归公式

$$Z=\mathrm{ite}(F,G,H)=\mathrm{ite}(v,\mathrm{ite}(F_v,G_v,H_v),\mathrm{ite}(F_{\overline{v}},G_{\overline{v}},H_{\overline{v}})) \tag{3}$$

式中 v 是任意变量。

证明 由式(2)可得

$$Z=vF_v+\overline{v}F_{\overline{v}}$$

又因为

$$Z=\mathrm{ite}(F,G,H)=FG+\overline{F}H$$

所以 $Z=v(FG+\overline{F}H)_v+\overline{v}(FG+\overline{F}H)_{\overline{v}}=v(F_vG_v+\overline{F}_vH_v)+\overline{v}(F_{\overline{v}}G_{\overline{v}}+\overline{F}_{\overline{v}}H_{\overline{v}})=$

1998 年 12 月 29 日收稿
* 国家科委重点科研项目
** 男 28 岁 博士生

$$\text{ite}(v, F_v G_v + \overline{F}_v H_v, F_{\bar{v}} G_{\bar{v}} + \overline{F}_{\bar{v}} H_{\bar{v}}) = \text{ite}(v, \text{ite}(F_v, G_v, H_v), \text{ite}(F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}}))$$　　证毕

利用式(3)计算 Z 的 BDD 存在两个难点，一是求 F，G 的香农展开因子，二是在通过递归方法求出 $\text{ite}(F_v, G_v, H_v)$ 和 $\text{ite}(F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}})$ 的结果 BDD 后，由 $\text{ite}(v, \text{ite}(F_v, G_v, H_v), \text{ite}(F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}}))$ 生成相应的 BDD 图。

假设 $F = (v, G, H)$，即根节点的判决变量为 v，其 1 子节点为 G，0 子节点为 H。一般情况下，计算 F_w 和 $F_{\bar{w}}$ 比较复杂，至多要遍历全部 BDD 节点。但在假定条件 $w \leqslant v$ 下，计算却很简单。当 $w < v$ 时：$F_w = F_{\bar{w}} = F$。当 $w = v$ 时：$F_w = G$；$F_{\bar{w}} = H$。

将 $w \leqslant v$ 称为快速香农展开条件，在 ite 迭代运算中很容易得到满足。假设 F，G，H 的根节点的判决变量分别为 v_1，v_2，v_3，选取 $v \leqslant v_i$，$i \in \{1,2,3\}$。这种情况满足快速香农的展开条件，所以，计算 F_v, G_v, H_v 和 $F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}}$ 十分容易，然后计算 $\text{ite}(F_v, G_v, H_v), \text{ite}(F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}})$。再由式(3)构造出 Z 的 BDD，见图 1。图中 $T = \text{ite}(F_v, G_v, H_v)$，$E = \text{ite}(F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}})$ 于是得出以下定理。

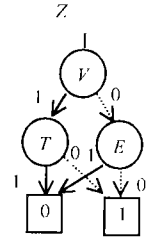

图 1　构造 Z 的 BDD

定理 1　函数 f_1，f_2，f_3 的 BDD 分别为 G_1，G_2，G_3，如果 G_1，G_2，G_3 中节点判决变量序互不交叉且 G_1 的节点编序最小，则 ite (G_1，G_2，G_3)可以按如下方法构成：将 G_1 中所有指向终节点 1 的边重定向到 G_2 根节点，将 G_1 中指向终节点 0 的边重定向到 G_3 的根节点。

从上面的分析可知，ite 算符能有效地使用递归进行运算，条件是选择 v，使 v 为 F、G、H 中节点序的最小者，即 F、G、H 中根节点的序的最小者。此时，ite 迭代公式可以表示为

$$\text{ite}(F, G, H) = (v, \text{ite}(F_v, G_v, H_v), \text{ite}(F_{\bar{v}}, G_{\bar{v}}, H_{\bar{v}})) \tag{4}$$

根据 ite 的定义，得到这个迭代公式的结束条件为

$$\text{ite}(F,1,0) = \text{ite}(1, F, G) = \text{ite}(0, G, F) = F \tag{5}$$

ite 迭代公式是整个 BDD 运算包的核心，由于它直接对 BDD 进行运算，并且能够实现所有的一元和二元逻辑运算，其他对 BDD 的操作也可以有效地建立在 ite 算符的基础上。

2　BDD 包的算法

BDD 包的核心 ite 算法的流程如图 2 所示：图中第 2 行中的结束条件由式(5)确定；第 3 行表示每次 ite 迭代都选择 F、G、H 中判决变量最小的一个来计算；第 6 行中的判断保证了 BDD 中没有冗余节点。第 7 行是在节点表中查找是否有节点(v, T, E)，如果存在，直接返回该节点的指针，如果不存在，则首先产生该节点，然后再加入节点表中，返回该节点指针。这就保证了节点表中没有同构的子图，从而保证所构造的 BDD 是正则的。

```
ite ( F, G, H){
if(满足结束条件)return 结果;
v= min (F→index, G→index, H→index);
T= ite (F_v, G_v, H_v);
E = ite(F_v̄, G_v̄, H_v̄);
if(T==E) return T;
R= findOrAddNodeTable(v,T,E);
return R;
}
```
图 2　ite 算法流程

构造电路的 BDD 时，首先读入电路的描述信息，确定变量顺序，从第一个输出开始构造其 BDD。在构造过程中，递归调用 ite 算法流程，进行深度优先搜索，然后逐级返回。如此循环，直至最后一个输出变量。

3　BDD 包的性能改进方法

直接采用上述的 ite 算法流程进行运算是不适用的，在最坏的情况下，该方法的时间复杂度为 $O(2^n)$(n 为输入变量个数)。我们对整个流程引入下面的改进方法。

```
typedef TagNode
{int index,
TagNode*  p1, p0;
} rawNode,
```

图3 BDD节点数据结构

3.1 哈希表

BDD是由许多节点构成的，每个节点的数据结构如图3所示。其中，index是节点 v 的判决变量编序，$p1$ 和 $p2$ 是指向子节点 high(v)和 low(v)的指针。终节点1和0的结构分别为：

1：{index=MAXINDEX+1；$p1$=1；$p0$=Null；}

0：{index=MAXINDEX+1；$p1$=Null；$p0$=1；}

在产生一个新的BDD节点之前，首先在BDD节点表中查找，如果找到，则直接返回节点指针；否则就产生新的BDD节点，并且加入到BDD节点表中。为了提高查找效率，我们引入了哈希方法。它通过哈希函数将元素的关键值映射到哈希表中。运用哈希表首先是使用哈希函数，将查找元素的关键值映射为哈希表中的地址。由于在实际应用中没有完美的哈希函数，多个不同元素可能映射到相同的地址，这就产生了冲突。因此，运用哈希表的第二步就是解决冲突。

```
typedef TagNode
{int index,
TagNode*  p1, p0;
TagNode*  next
}uniqueNode,
```

图4 修改后的BDD节点数据结构

根据BDD的数据结构，我们选择(index, $p0$, $p1$)作为一个BDD节点的关键值，采用的哈希函数为

$$\text{hash}(\text{index}, p0, p1)=(\text{index} + 3p1 + 5p0)\%T$$

式中 T 为哈希表的大小，对于有相同哈希值的多个元素，采用链表的方式连接起来。考虑到哈希链表的应用，修改BDD节点的数据结构见图4。指针next指向具有相同哈希值的下一个元素。通过哈希表，能够实现元素的快速查找，并且保证所产生的BDD节点具有唯一性。

3.2 属性边

使用反向属性边，即在指向BDD节点的边上增加一个属性值表示逻辑取非，这样，函数 f 和 $\overline{f}$ 就由一个节点表示，只是指向该节点的边的属性相反。反向属性边的引入影响了BDD的规范性，采用下面两条规则保证进行修正：1) 每个节点的1边取反属性不能为真；2) BDD包中只有一个终节点1。

由于BDD运算包中反向属性边由引用指针的最低位来表示，所以引入反向属性边前后每个节点所占用的内存总数是相等的。

3.3 提高 ite 运算效率的规则

为了在有限的记忆容量下尽可能提高命中率，在BDD包中引入了以下几条规则来对参加运算的函数进行标准化处理。这样，在不增加缓存的内存用量的前提下，减少了记忆覆盖的概率，相应地提高了缓存命中率。

1) 化简规则

$$\text{ite}(F,F,G)=\text{ite}(F,1,G) \qquad \text{ite}(F,G,F)=\text{ite}(F,G,0)$$
$$\text{ite}(F,G,\overline{F})=\text{ite}(F,G,1) \qquad \text{ite}(F,\overline{F},G)=\text{ite}(F,0,G)$$

2) 等效规则

由于二元操作的可交换性，例如 $fg = gf$，$f + g = g + f$，在用ite进行运算时，就在等效的表达式中选择一种，使第一个参数的根节点的判决变量的序最小。如果两个参数的根节点的判决变量的序相等，则第一个参数的内存地址必须最小。

3) 反向属性边规则

ite的三个参数中，前两个参数的反向属性边不能为真，可通过下面的等效转换来实现

$$\text{ite}(F,G,H)=\text{ite}(\overline{F},H,G)=\overline{\text{ite}(F,\overline{G},\overline{H})}=\overline{\text{ite}(\overline{F},\overline{H},\overline{G})}$$

采用这些标准化技术，可以检查到满足德摩根定律的等效形式，如 $A+B=\mathrm{ite}(A,1,B)$ 和 $\overline{A}\cdot\overline{B}=\mathrm{ite}(\overline{A},\overline{B},0)=\overline{\mathrm{ite}(A,1,B)}$，这归功于 ite 算符将各种二元运算以一种统一的形式进行和 BDD 包中的反向属性边。

4 结　论

在前面所述的 ite 算法的基础上，我们在 PC 平台(Pentium 120 MHz，32 MRAM，Windows95)上以 C++语言实现了一个高效通用的 BDD 运算包。为了验证该 BDD 包的性能，我们构造了 ISCAS'85 标准电路的 BDD，如表 1 所示。表中同时列出了 K.S.Brace 等人的 CMU BDD 包在近似条件下构造的 BDD 的结果[3]，试验是在 Sun 工作站上完成的。

表 1　BDD 总体性能比较

电路名	时间/s		节点数	
	本文结果	文献[2]	本文结果	文献[2]
c432	5.61	79.5	31 178	30 200
c499	31.15	85.0	40 658	49 786
c880	0.82	10.5	7 279	7 655
c1355	79.09	80.4	40 658	39 858
c1908	2.14	28.9	12 712	12 463
c2670	1.70	60.4	46 194	18 153
c3540	35.16	704.0	137 530	208 947
c5315	1.65	51.0	21 383	32 193
c7552	14.12	60.5	44 604	5 895

ISCAS'85 的 10 个标准电路中，c6288 是一个乘法器，是公认不能用 BDD 处理的，因此表 1 中没有列出。由于输入变量的编序方法不同，表中列出的节点数只具有参考价值。但可以看出，在产生的 BDD 节点数近似相等的情况下，本文 BDD 包的运算速度更快。

BDD 包的成功实现，为 BDD 的应用打下了坚实的基础，使得对 BDD 的进一步研究成为可能。

参 考 文 献

1　Bryant Randal E. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers. 1986, c-35 (8) : 677~691

2　李翔宇，陈光禹．二元判决图变量排序新方法.电子科技大学学报，1999，28（2）：152~156

3　Brace Karl S, Rudell Richard L, Bryant Randal E. Efficient implementation of a BDD package. 27th ACM/IEEE Design Automation Conference, 1990 : 40~45

4　Minato Shin-ichi, Ishiura Nagisa, Yajima Shuzo. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. 27th ACM/IEEE DAC, 1990 : 52~57

Realization and Improvement of Binary Decision Diagram

Li Xiangyu　　Chen Guangju

(CAT Lab., UEST of China　Chengdu　610054)

Abstract　If-then-else (ite) functor is the core of constructing binary decision diagram (BDD) for a Boolean function. This paper discusses how to express the fundamental operations of Boolean function in ite functor, and how to construct BDD utilizing ite functor. Meanwhile, some methods, such as memory function, hash table, complement attributed edge, are used to improve the efficiency of the package.

Key words　binary decision diagram;　ite functor;　hash table;　complement attributed edge