第一章　面向流控机制的对偶综合

1.1

在通讯和多媒体芯片设计中，一个最为困难且容易出错的工作就是设计该协议的的编码器和解码器。其中编码器将输入变量 $\vec{i}$ 映射到输出变量 $\vec{o}$. 而解码器则从 $\vec{o}$ 中恢复 $\vec{i}$。对偶综合算法 [1–7] 通过自动生成特定编码器的解码器以降低该工作的复杂度并提高结果的可靠性。该算法的一个前提假设是，编码器的输入变量 $\vec{i}$ 总能够被输出变量 $\vec{o}$ 的一个有限长度序列唯一决定。

然而，许多高速通讯系统的编码器带有流控功能[8]，而该功能直接违反了上述假设。图1.1a) 展示了一个带有流控机制的通讯系统的结构。其中一个传输器 (transmitter) 和一个接收器 (receiver) 通过一个编码器 (encoder) 和一个解码器 (decoder) 连接在一起。从传输器到编码器有两个输入变量：有待编码的数据位 d，和代表 d 的有效性的有效位 f。图1.1b) 给出了编码器如何将 f 和 d 映射到输出变量 $\vec{o}$ 的编码表。

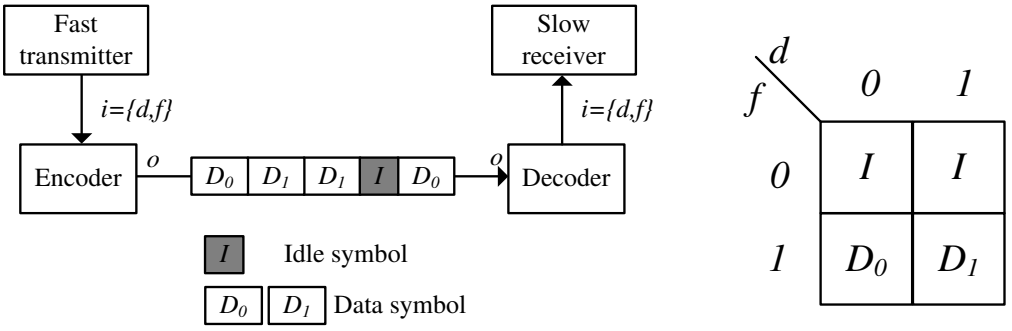
流控机制的工作原理为：

1. 当接收器能够跟上发送器的速度时，发送器将 f 设为 1，这使得编码器按照 d 的值发送 D_d。从图1.1b) 的编码表可知，解码器总能够根据 D_d 恢复 f 和 d。

2. 而当接收器无法跟上发送器的速度时，发送器将 f 设为 0 以阻止编码器继续发送 D_d，转而在不考虑 d 的情况下发送空闲符号 I。而解码器应当识别并淘汰 I，并将 $f \equiv 0$ 发送给接收器。此时 d 的具体值并不重要。

上述流控机制能够防止快速发送器发送过多数据以至于接收器无法处理。然而该机制违反了迄今为止所有对偶综合算法 [1–7] 的基本假设，因为 d 无法被 I 唯一决定。很明显，为了解决该问题，我们只需考虑 $f \equiv 1$ 的情形，因为在此情况下 d 是能够被唯一决定的。而对于 $f \equiv 0$ 的情况，d 是无意义的，可以无需考虑。

基于上述讨论，本文提出了首个能够处理流控机制的对偶综合算法。该算法分为三步：**首先**, 它使用经典的对偶综合算法[3] 以识别那些能够被唯一决定的输入变量，并称他们为流控变量 $\vec{f}$。而其他不能被唯一决定的变量称为数据变量 $\vec{d}$。**第二**，该算法推导一个充分必要谓词 $valid(\vec{f})$ 使得 $\vec{d}$ 能够被输出变量 $\vec{o}$ 的一个有限长度序列唯一决定。**第三**, 对于每一个流控变量 $f \in \vec{f}$，该算法使用 Craig 插值算法[9] 特征化其解码器函数。同时，对于数据变量 $\vec{d}$，他们的值只有在 $valid(\vec{f}) \equiv 1$



a) The faster transmitter and slower receiver connected by the encoder and decoder

b) The mapping table of the encoder with flow control mechanism that maps d and f to o

图 1.1　带有流控机制的通讯系统及其编码器

时才有意义。因此每个 $d \in \vec{d}$ 的解码器函数可以类似的使用 Craig 插值算法得到，唯一的不同在于必须首先应用谓词 $valid(\vec{f}) \equiv 1$。

该算法的第二步类似于[4]，因为他们都试图得到使得 $\vec{d}$ 或者 $\vec{i}$ 被唯一决定的谓词。然而他们的根本区别在于 [4] 的算法推导的是一个全局谓词，必须在整个展开的迁移关系上都成立。而本文算法得到的是仅应用于当前步的谓词，以恢复 $\vec{d}$。因此本文算法可以视为[4] 的一般化。

实验结果表明, 对于多个来自于工业界的复杂编码器 (如以太网 [10] 和 PCI Express [11])，本文算法总能够正确的识别流控变量 $\vec{f}$，推导谓词 $valid(\vec{f})$, 并产生解码器。同时，我们也和现有算法进行了运行时间，电路面积和时序方面的比较。

本章剩余部分按照以下方式组织。第1.2节介绍了背景知识；第1.3节给出了识别流控变量的算法，而第1.4节推导使得 $\vec{d}$ 能够被 $\vec{o}$ 唯一决定的谓词；第1.5节压缩迁移关系展开的长度以降低运行时间并减小电路面积和延时，而第1.6节给出了特征化解码器电路的算法；第1.7和1.8节分别给出了实验结果和相关工作。最后，第1.9节给出了结论。

1.2

本节介绍相关的背景知识。

1.2.1　命题逻辑可满足问题

布尔集合记为 $B = \{0, 1\}$。多个变量组成的向量记为 $\vec{v} = (v, \ldots)$。$\vec{v}$ 中的变量个数记为 $|\vec{v}|$。如果 v 是 $\vec{v}$ 的成员，则记为 $v \in \vec{v}$；否则 $v \notin \vec{v}$。对于变量 v 和向量 $\vec{v}$，如果 $v \notin \vec{v}$，则同时包含 v 和所有 $\vec{v}$ 的成员的新向量记为 $v \cup \vec{v}$。如果 $v \in \vec{v}$，则包含所有 $\vec{v}$ 的成员而不包含 v 的新向量记为 $\vec{v} - v$。对于两个向量 $\vec{a}$ 和 $\vec{b}$，包含 $\vec{a}$ 和 $\vec{b}$ 的所有成员的新向量记为 $\vec{a} \cup \vec{b}$。向量 $\vec{v}$ 的赋值集合记为 $[\![\vec{v}]\!]$。例如：$[\![(v_1, v_2)]\!] = \{(0,0), (0,1), (1,0), (1,1)\}$。

在变量集合 V 上的布尔逻辑公式 F 是通过以下连接符连接 V 上的变量得到的：包括 $\neg, \wedge, \vee$ 和 $\Rightarrow$，他们分别代表着求反，与，或和蕴含操作。

对在变量集合 V 上的布尔逻辑公式 F，命题逻辑可满足问题 (缩写为 SAT) 意味着寻找 V 的赋值函数 $A : V \rightarrow B$，使得 F 可以取值为 1。如果存在这样的赋值函数 A，则 F 是可满足的；否则，是不可满足的。

一个寻找上述赋值函数 A 的计算机程序称为 SAT 求解器，常见的 SAT 求解器包括 Zchaff[12]，Grasp[13]，Berkmin[14]，和 MiniSat[15]。

通常，SAT 求解器要求有待求解的公式使用合取范式 (CNF) 表示，其中一个公式是一个短句集合的合取，一个短句是一个文字集合的析取，一个文字是一个变量或者其反。一个使用 CNF 格式表示的公式通常也称为 SAT 实例。

从文献[16] 可知，对于函数 $f(v_1 \ldots v \ldots v_n)$，针对变量 v 的正 cofactor 和负 cofactor 分别是 $f_{v\equiv 1} = f(v_1 \ldots 1 \ldots v_n)$ 和 $f_{v\equiv 0} = f(v_1 \ldots 0 \ldots v_n)$。而 Cofactoring 则代表着将 1 或者 0 赋予 v 以得到 $f_{v\equiv 1}$ 和 $f_{v\equiv 0}$。

给定两个布尔逻辑公式 ϕ_A 和 ϕ_B，若 $\phi_A \wedge \phi_B$ 不可满足，则存在仅使用了 ϕ_A 和 ϕ_B 共同变量的公式 ϕ_I ，使得 $\phi_A \Rightarrow \phi_I$ 且 $\phi_I \wedge \phi_B$ 不可满足。ϕ_I 被称为 ϕ_A 针对 ϕ_B 的 Craig 插值[17]。ϕ_I 可以使用 McMillan 算法[9] 得到。Craig 插值通常被用于产生 ϕ_A 的上估计。

1.2.2 MiniSat 求解器的递增求解机制

本文中，我们使用 MiniSat 求解器[15] 求解所有 CNF 公式。和其他基于冲突学习机制[18] 的 SAT 求解器类似，MiniSat 从在搜索中遇到的冲突中产生学习短句，并记录他们以避免类似的冲突再次出现。该机制能够极大的提升 SAT 求解器的性能。

在许多应用中，经常存在一系列紧密关联的 CNF 公式。如果在一个 CNF 公式求解过程中得到的学习短句能够被其他 CNF 公式共享，则所有 CNF 公式的求解速度都能够得到极大的提升。

MiniSat 提供了一个增量求解机制以共享这些学习短句。该机制包括两个接口函数：

1. *addClause*(F) 用于将一个 CNF 公式 F 添加到 MiniSat 的短句数据库，以用于下一轮求解。

2. *solve*(A) 接收一个文字集合 A 作为假设，并求解 CNF 公式 $F \wedge \bigwedge_{a\in A} a$。其中 F 是在 *addClause* 中被加入短句数据库的 CNF 公式。

1.2.3 有限状态机

本文中编码器使用有限状态机 $M = (\vec{s}, \vec{i}, \vec{o}, T)$ 作为模型。一个状态机包括状态变量向量 $\vec{s}$，输入变量向量 $\vec{i}$，输出变量向量 $\vec{o}$，和迁移函数 $T : [\![\vec{s}]\!] \times [\![\vec{i}]\!] \to [\![\vec{s}]\!] \times [\![\vec{o}]\!]$。其中 T 用于从当前状态变量向量 $\vec{s}$ 和输入变量向量 $\vec{i}$ 计算出下一状态变量向量 $\vec{s}$ 和输出变量向量 $\vec{o}$。

有限状态机 M 的行为可以通过将迁移函数展开多步得到。在第 n 步上，状态变量 $s \in \vec{s}$，输入变量 $i \in \vec{i}$ 和输出变量 $o \in \vec{o}$ 分别表示为 s_n，i_n 和 o_n。进一步的，在第 n 步的状态变量向量，输入变量向量和输出变量向量分别记为 $\vec{s}_n$，$\vec{i}_n$ 和 $\vec{o}_n$。一条路径是一个状态序列 $< \vec{s}_n, \ldots, \vec{s}_m >$ 使得 $\exists \vec{i}_j \vec{o}_j (\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$ 对所有 $n \le j < m$ 均满足。一个环是一条路径 $< \vec{s}_n, \ldots, \vec{s}_m >$ 使得 $\vec{s}_n \equiv \vec{s}_m$。

1.2.4 确认一个输入变量是否能够被输出变量的有限长序列唯一决定的停机算法

目前所有的对偶综合算法 [1–7] 均假设 $\vec{i}$ 能够被输出变量的有限长序列唯一决定，因此他们均将 $\vec{i}$ 视为一个整体，而不单独考虑每个 $i \in \vec{i}$。然而在本文中，我们需要检查每一个单独的 $i \in \vec{i}$。因此本文的表达方式有可能和现有算法 [1–7] 存在细微差别。

我们在文献[3] 中提出了第一个完成此任务的停机算法。其基本思想是将迁移函数展开至递增的长度序列。而对于每一个展开长度，使用两个近似估计算法以获得当前的答案。第一个估计算法是第1.2.4.1节中描述的下估计算法。而第二个估计算法是第1.2.4.2节中描述的上估计算法。因此，当第一个算法给出的答案是” 是” 的时候，则最终答案也是” 是”，当第二个算法给出的答案是” 否” 的时候，则最终答案也是” 否”。我们将在第1.2.4.3 节中证明这两个算法随着展开长度的增长必将收敛并停机，并给出确定的答案。

1.2.4.1

如图1.2所示，在展开的迁移函数序列上，如果存在三个参数 p, l 和 r，使得对于输出序列 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 的任意取值，i_{p+l} 不能同时取值为 0 和 1，则输入变量 $i \in \vec{i}$ 可以被唯一决定。这等价于等式 (1.1) 中的公式 $F_{PC}(p,l,r)$ 的不可满足。

$$F_{PC}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.1}$$

这里，p 是前置迁移函数序列的长度，l 和 r 则是两个用于唯一决定 i_{p+l} 的输出序列 $<\vec{o}_{p+1},\ldots,\vec{o}_{p+l}>$ 和 $<\vec{o}_{p+l+1},\ldots,\vec{o}_{p+l+r}>$ 的长度。等式 (1.1) 的行 1 对应于图1.2左边的路径，而行 2 对应于图1.2右边的路径。他们的长度是相同的。行 3 强制这两条路径的输出是相同的。而行 4 要求他们的输入 i_{p+l} 是不同的。行 5 和 6 则是用户给出的断言，用于约束合法的输入模式。F_{PC} 中的 PC 是"parameterized complementary" 的缩写，意味着 $F_{PC}(p,l,r)$ 被用于检查在三个参数 p,l 和 r 的情况下，i_{p+l} 能否被唯一决定。

从图1.2可知，等式 (1.1) 的前三行代表了两个具有相同输出的迁移函数展开序列。因此他们总是可满足的。而最后两行是对合法输入模式的约束。我们将在算法开始前检查他们的可满足性。所以 $F_{PC}(p,l,r)$ 的不可满足意味着 $i_{p+l} \equiv i'_{p+l}$，即输入被唯一决定。

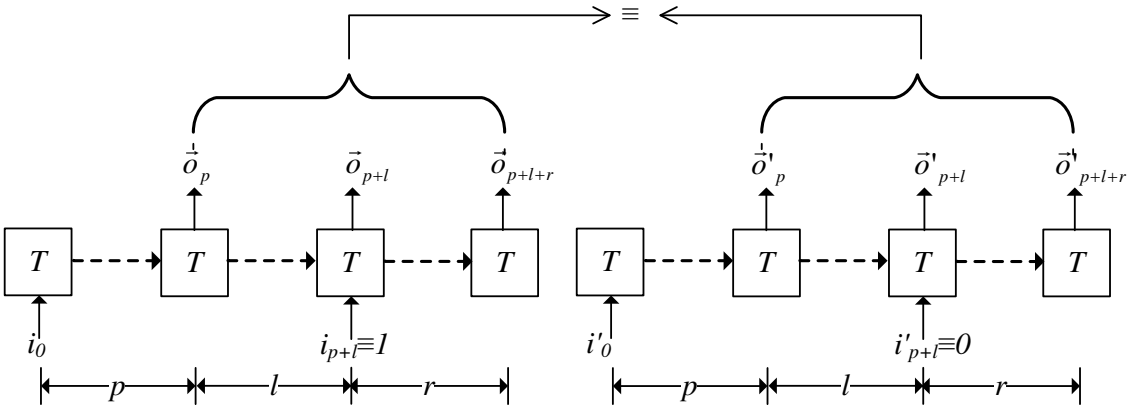


图 1.2　用于检查 i_{p+l} 是否能够被唯一决定的下估计算法

从图1.2可知，如果 $F_{PC}(p,l,r)$ 不可满足则 $F_{PC}(p',l',r')$ 对于更大的 $p' \geq p$, $l' \geq l$ 和 $r' \geq r$ 也不可满足。从等式 (1.1) 中可知，$F_{PC}(p',l',r')$ 的短句集合是 $F_{PC}(p,l,r)$ 的超集。这也指向了同一个结论。

这意味着，$F_{PC}(p,l,r)$ 的不可满足性的限界证明可以被扩展到任意 p, l 和 r 上，从而成为非限界的证明。

命题 1.1: 如果 $F_{PC}(p,l,r)$ 不可满足则对于任意更大的 p, l 和 r，i_{p+l} 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

等式 (1.1) 不包含初始状态，相反使用一个长度为 p 步的前置状态序列 $< \vec{s_0}, \ldots, \vec{s_{p-1}} >$ 以将约束 $assertion(\vec{i})$ 传播到状态序列 $< \vec{s_p}, \ldots, \vec{s_{p+l+r}} >$。从而将在 $assertion(\vec{i})$ 约束下不可达的状态集合剔除。相比考虑初始状态的传统方法，这带来了两个主要的好处：首先，通过不计算可达状态，本文算法可以得到极大的简化和加速。而相比之下，目前唯一能够计算可达状态的对偶综合算法[7] 则无法处理最为复杂的 XFI 编码器。而我们的算法[3] 则始终可以处理。第二，通过忽略初始状态，本文算法可以提升产生出来的解码器的可靠性。因为这可以使得解码器的状态和输出仅仅依赖于有限的输入历史。因此任何被传输中的错误破坏的 $\vec{o}$ 只能对解码器产生有限步数的影响。

当然忽略初始状态有一个缺点在于，他使得判断条件比必须的情况稍微强一些。也就是说，他要求 $\vec{i}$ 必须在一个更大的集合 R^p 上被唯一决定。其中 R^p 代表了由任意状态在 p 步之内能够到达的状态集合。而必要条件是从初始条件出发在任意步数内可以到达的可达状态集合 R。因此在某些情况下，我们的算法有可能无法处理正确设计的编码器。不过从目前使用的所有编码器，即使是那些来自于真实工业应用的编码器，这种极端情况也没有出现过。

1.2.4.2

.

在上一小节，我讨论了当 $F_{PC}(p,l,r)$ 不可满足时，i_{p+l} 能够针对任意 p, l 和 r 被唯一决定。另一方面，如果 $F_{PC}(p,l,r)$ 是可满足的，则 i_{p+l} 不能在特定的 p, l 和 r 情况下被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。此时存在两种可能性：

1. 在更大的 p, l 和 r 情况下 i_{p+l} 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

2. 对任意 p, l 和 r，i_{p+l} 都不能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

如果是第一种情况，则通过迭代的增长 p, l 和 r，$F_{PC}(p,l,r)$ 总能够变成不可满足。然而对于第二种情况，迭代的递增 p, l 和 r 将导致不停机。

因此，为了得到一个停机算法，我们需要区分上述两种情形的手段。该手段如图1.3所示，该图类似于图1.2，但是增加了三个约束用于检测三个路径 $<\vec{s}_0,\ldots,\vec{s}_p>$,$<\vec{s}_{p+1},\ldots,\vec{s}_{p+l}>$ 和 $<\vec{s}_{p+l+1},\ldots,\vec{s}_{p+l+r}>$ 上的环。该方法被形式化的定义于等式 (1.2) 中。

$$F_{LN}(p,l,r) := \left\{ \begin{array}{ll} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{1.2}$$

F_{LN} 中的 LN 意味着环形非对偶。这表明 $F_{LN}(p,l,r)$ 将使用这三个环检测约束来检测 i_{p+l} 是否不能够被唯一决定。

当 $F_{LN}(p,l,r)$ 可满足，则 i_{p+l} 不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。更重要的，通过展开这三个环，我们能将这个结论扩展到任意更大的 p,l 和 r 上。这意味着:

命题 1.2: 当 $F_{LN}(p,l,r)$ 可满足时，i_{p+l} 针对任意 p,l 和 r 都不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。

1.2.4.3

.

根据命题1.1 和1.2, 我们能将针对特定 p,l 和 r 的限界证明扩展到针对任意 p,l 和 r 的非限界情形。这使得我们得到停机算法1.1，用于检测 $i \in \vec{i}$ 是否能被 $\vec{o}$ 的有限长度序列唯一决定。

1. 一方面，如果确实存在 p,l 和 r，使得输入能被输出唯一决定，令 $p' := max(p,l,r), l' := max(p,l,r)$ 和 $r' := max(p,l,r)$。从命题1.1，可知 $F_{PC}(p',l',r')$ 是不可满足的。因此 $F_{PC}(p,l,r)$ 总能够在算法1.1行4成为不可满足的并退出循环;

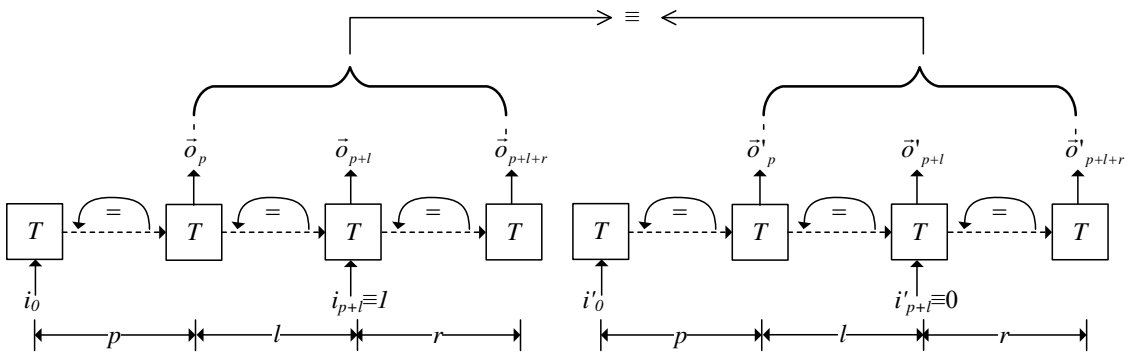


图 1.3　用于检查 i_{p+l} 是否不能被唯一决定的上估计算法

算法 1.1 *CheckUniqueness*(*i*): 用于检测 $i \in \vec{I}$ 是否能够被 $\vec{o}$ 的有限长度序列唯一决定的停机算法

```
1: p:= 0; l:= 0; r:= 0
2: while 1 do
3:         p++; l++; r++ ;
4:         if F_PC(p,l,r) 不可满足 then
5:                 return (1,p,l,r);
6:         else if F_LN(p,l,r) 可满足 then
7:                 return (0,p,l,r);
8:         end if
9: end while
```

2. 另一方面，如果不存在这样的 p, l 和 r，则 p, l 和 r 在不断的递增之后最终总能够大于有限状态机的最大无环路径长度。这意味着在 $< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ 和 $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$ 上都存在环。这将使得 $F_{LN}(p,l,r)$ 在行6被满足。这样将导致退出循环。

因此该算法是停机的。

1.3

本节将介绍如何识别流控变量 $\vec{f}$。这将首先在小节1.3.1中介绍。然后小节1.3.2将介绍如何使用增量求解算法加速该算法。

1.3.1 识别流控变量 $\vec{f}$

为了便于描述，我们假设 $\vec{I}$ 可以被划分为两个向量: 流控向量 $\vec{f}$ 和数据向量 $\vec{d}$。

流控向量 $\vec{f}$ 用于表达 $\vec{d}$ 的有效性。所以对于一个正确设计的编码器，$\vec{f}$ 总能够被输出变量向量 $\vec{o}$ 的一个有限长度序列唯一决定。否则解码器无法识别 $\vec{d}$ 的有效性。

所以我们提出算法1.2用于识别 $\vec{f}$。

在行1，f 和 d 被设为空向量。在行2，p, l 和 r 的初始值被设为 0。

在行3，一个 while 循环被用于遍历 $i \in \vec{I}$。

在行6，能够被唯一决定的输入变量 i 将被加入 $\vec{f}$。

另一方面，当 $\vec{I}$ 非常长时，逐一测试 $i \in \vec{I}$ 的时间开销将会很大。为了加速该算法，当 $F_{LN}(p,l,r)$ 在行9是可满足的时候，每个在 j_{p+l} 和 j'_{p+l} 上取不同值的 $j \in \vec{I}$ 也将被在行10加入 $\vec{d}$，因为他们自己的 $F_{LN}(p,l,r)$ 也是可满足的。

在某些特殊情况下，一些 $d \in \vec{d}$ 也能够像 $f \in \vec{f}$ 那样被唯一决定。在这种情况下，d 也将被算法1.2识别为流控变量。但是这不会对我们的算法产生不利影响，因为这些 d 的解码器函数也将在节1.6中被正确特征化。

1.3.2 使用增量 SAT 求解加速识别算法

通过节1.2.2中的 MiniSat 增量求解机制，我们能进一步加速算法1.2 。

通过将等式 (1.1) 的第三行移动到等式 (1.4)，等式 (1.1) 中的 $F_{PC}(p,l,r)$ 可以被划分为以下两个等式：

$$C_{PC}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.3}$$

$$A_{PC}(p,l,r) := \left\{ \; i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \; \right\} \tag{1.4}$$

算法 1.2 *FindFlow*($\vec{i}$): 识别 $\vec{f}$

```
1:  f⃗ := {}; d⃗ := {};
2:  p := 0 ; l := 0 ; r := 0 ;
3:  while i⃗ ≠ {} do
4:      假设 i ∈ i⃗;
5:      p++;  l++;  r++;
6:      if F_PC(p,l,r) 对于 i 不可满足 then
7:          f⃗ := i ∪ f⃗;
8:          i⃗ := i⃗ − i;
9:      else if F_LN(p,l,r) 对于 i 可满足且赋值为 A then
10:         for j ∈ i⃗ do
11:             if A(j_{p+l}) ≠ A(j'_{p+l}) then
12:                 i⃗ := i⃗ − j;
13:                 d⃗ := j ∪ d⃗;
14:             end if
15:         end for
16:     end if
17: end while
18: return (f⃗,p,l,r);
```

类似的我们可以将等式 (1.2) 中的 $F_{LN}(p,l,r)$ 划分为下列两个等式:

$$C_{LN}(p,l,r) := \left\{ \begin{array}{ll} & C_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{1.5}$$

$$A_{LN}(p,l,r) := \left\{ \; i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \; \right\} \tag{1.6}$$

算法 1.3 *FindFlowIncSAT*($\vec{i}$): 基于增量求解识别流控变量

```
1:  f⃗ := {}; d⃗ := {} ;
2:  p := 0 ; l := 0 ; r := 0 ;
3:
4:  while i⃗ ≠ {} do
5:      p++; l++; r++;
6:      addClause(C_PC(p,l,r));
7:      for i ∈ i⃗ do
8:          if solve(A_PC(p,l,r)) 不可满足 i then
9:              i⃗ := i⃗ − i;
10:             f⃗ := i ∪ f⃗;
11:         end if
12:     end for
13:     addClause(C_LN(p,l,r));
14:     for i ∈ i⃗ do
15:         if solve(A_LN(p,l,r)) 可满足 i 且赋值为 A then
16:             for j ∈ i⃗ do
17:                 if A(j_{p+l}) ≠ A(j'_{p+l}) then
18:                     i⃗ := i⃗ − j;
19:                     d⃗ := j ∪ d⃗;
20:                 end if
21:             end for
22:         end if
23:     end for
24: end while
25: return (f⃗,p,l,r)
```

很明显 C_{PC} 和 C_{LN} 是独立于特定 $i \in \vec{i}$ 的，所以他们可以使用 $addClause(C_{PC})$ 或 $addClause(C_{LN})$ 来被添加进入 MiniSat 的短剧数据库。于此同时，A_{PC} 和 A_{LN} 中的短句只包含文字，因此他们可以作为调用 *solve* 函数时的假设集合。

因此，基于上述等式，我们可以使用增量求解机制将算法1.2 修改成为算法1.3 。主要的修改在于行6 和13上的两个 *addClause* ，以及行8 和15上的两个 *solve*。他们是在小节1.2.2中描述的 MiniSat 的增量求解机制接口。

1.4

本节我们将描述使得数据变量 $\vec{d}$ 向量能够被唯一决定的谓词 $valid(\vec{f})$。在小节1.4.1中，我们提出了一个用于特征化使得某个特定 CNF 公式被满足的条件的算法。在小节1.4.2中，我们使用上述算法推导 $valid(\vec{f})$, 也就是使得 $\vec{d}$ 能够被 $\vec{o}$ 的有限长度序列唯一决定的谓词。

1.4.1 特征化使得特定 CNF 公式被满足的谓词

假设 $R(\vec{a},\vec{b},t)$ 是一个使得 $R(\vec{a},\vec{b},0) \wedge R(\vec{a},\vec{b},1)$ 不可满足的布尔公式。

其中 $\vec{a}$ 和 $\vec{b}$ 被分别称为重要和非重要变量子集。而 t 是目标变量。我们进一步假设 $R(\vec{a},\vec{b},t)$ 是可满足的。

我们需要特征化一个布尔函数 $FSAT_R(\vec{a})$，覆盖且仅覆盖了所有能够使得 $R(\vec{a},\vec{b},1)$ 可满足的 $\vec{a}$。形式化的定义是：

$$FSAT_R(\vec{a}) := \begin{cases} 1 & \exists \vec{b}.R(\vec{a},\vec{b},1) \\ 0 & otherwise \end{cases} \tag{1.7}$$

因此，一个计算 $FSAT_R(\vec{a})$ 的简单算法是：逐一遍历并收集所有使得 $R(\vec{a},\vec{b},1)$ 可满足的 $\vec{a}$ 的赋值。然而该算法需要处理 $2^{|\vec{a}|}$ 中情况。对于很长的 $\vec{a}$，时间开销将会很大。

使用 cofactoring [16] 和 Craig 插值[9]，可以将每一个 $\vec{a}$ 扩展为一个更大的集合，从而极大的提高算法运行速度。直观的，假设 $R(\vec{a},\vec{b},1)$ 的一个满足赋值是 $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0,1\}$，通过 cofactoring[16] 可以构造以下公式：

$$R(\vec{a},A(\vec{b}),1) := R(\vec{a},\vec{b},1)_{b \equiv A(b)} \tag{1.8}$$

因为 $R(\vec{a},A(\vec{b}),0) \wedge R(\vec{a},A(\vec{b}),1)$ 是不可满足的，$R(\vec{a},A(\vec{b}),1)$ 针对 $R(\vec{a},A(\vec{b}),0)$ 的 Craig 插值 $ITP(\vec{a})$ 可以用作 $\vec{a}$ 使得 $R(\vec{a},A(\vec{b}),1)$ 可满足的上估计。同时，$ITP(\vec{a}) \wedge$

算法 1.4 *CharacterizingFormulaSAT*$(R,\vec{a},\vec{b},t)$: 特征化使得 $R(\vec{a},\vec{b},1)$ 可满足的 $\vec{a}$ 集合

1: $FSAT_R(\vec{a}) := 0$;
2: **while** $R(\vec{a},\vec{b},1) \wedge \neg FSAT_R(\vec{a})$ 是可满足的 **do**
3: 假设 $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0,1\}$ 可满足赋值函数;
4: $\phi_A(\vec{a}) := R(\vec{a}, A(\vec{b}), 1)$;
5: $\phi_B(\vec{a}) := R(\vec{a}, A(\vec{b}), 0)$;
6: 假设 $ITP(\vec{a})$ 是 ϕ_A 针对 ϕ_B 的 Craig 插值;
7: $FSAT_R(\vec{a}) := ITP(\vec{a}) \vee FSAT_R(\vec{a})$;
8: **end while**
9: **return** $FSAT_R(\vec{a})$

$R(\vec{a}, A(\vec{b}), 0)$ 是不可满足的，因此 $ITP(\vec{a})$ 没有覆盖任何使得 $R(\vec{a}, A(\vec{b}), 0)$ 可满足的情况。因此，$ITP(\vec{a})$ 覆盖且仅覆盖了所有使得 $R(\vec{a}, A(\vec{b}), 1)$ 可满足的 $\vec{a}$。

基于上述讨论，我们提出了算法1.4 以特征化等式 (1.7) 中的 $FSAT_R(\vec{a})$。行2检测是否仍然存在尚未被 $FSAT_R(\vec{a})$ 覆盖的 $\vec{a}$ ，使得 $R(\vec{a},\vec{b},1)$ 可满足。行4 和5 将可满足赋值中 $\vec{b}$ 的取值分别赋予 $R(\vec{a},\vec{b},1)$ 和 $R(\vec{a},\vec{b},0)$ 。这将使得 $\vec{b}$ 不在出现在这两个公式中。

因此，$\phi_A \wedge \phi_B$ 在行6 是不可满足的。且 ϕ_A 和 ϕ_B 的共同变量是 $\vec{a}$。因此可以使用 McMillian 算法[9] 计算 Craig 插值 $ITP(\vec{a})$。

$ITP(\vec{a})$ 将在行7被加入 $FSAT_R(\vec{a})$ 并在行2 被排除。

算法1.4 的每一个循环将向 $FSAT_R(\vec{a})$ 中加入至少一个 $\vec{a}$ 的赋值。这意味着 $FSAT_R(\vec{a})$ 覆盖了 $\vec{a}$ 的一个有界并且单调增长的赋值集合。因此算法1.4 是停机的。

1.4.2 推导使得 $\vec{d}$ 被唯一决定的谓词 $valid(\vec{f})$

本节中，我们将给出如何推导使得 $\vec{d}$ 被唯一决定的谓词 $valid(\vec{f})$。

如图1.4所示，我们将首先在小节1.4.2.1中定义 $\neg FSAT_{PC}(p,l,r)$，$valid(\vec{f})$ 的一个单调增长的下估计。然后我们将在小节1.4.2.2中定义 $\neg FSAT_{LN}(p,l,r)$，$valid(\vec{f})$ 的一个单调递减的上估计。然后在小节1.4.2.3中我们指出这两个估计将收敛至 $valid(\vec{f})$ 。小节1.4.3将证明该算法的正确性。

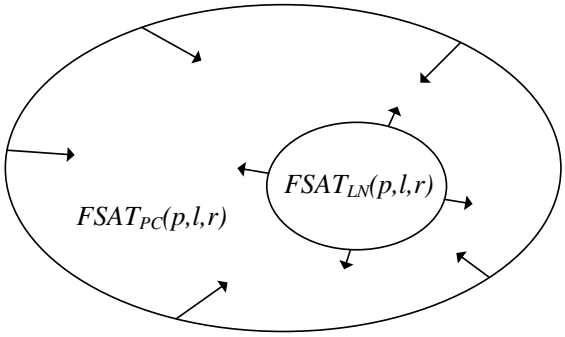


图 1.4　$FSAT_{PC}(p,l,r)$ 和 $FSAT_{LN}(p,l,r)$ 的单调性

1.4.2.1　$valid(\vec{f})$

通过将等式 (1.1) 中的 i 替换为 $\vec{d}$，我们得到：

$$F^{d}_{PC}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r}\vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.9}$$

如果 $F^{d}_{PC}(p,l,r)$ 是可满足的，则 $\vec{d}_{p+l}$ 无法被 $< \vec{o}_p,\ldots,\vec{o}_{p+l+r} >$ 唯一决定。通过收集等式 (1.9) 的第三行，我们得到 $T_{PC}(p,l,r)$：

$$T_{PC}(p,l,r) := \left\{ \bigwedge_{m=p}^{p+l+r}\vec{o}_m \equiv \vec{o'}_m \right\} \tag{1.10}$$

通过将 $T_{PC}(p,l,r)$ 代入到 $F^{d}_{PC}(p,l,r)$，我们得到一个新的公式：

$$F'^{d}_{PC}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{PC}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.11}$$

很明显 $F^{d}_{PC}(p,l,r)$ 和 $F'^{d}_{PC}(p,l,r,1)$ 是等价的。我们进一步定义：

$$\vec{a} := \vec{f}_{p+l} \tag{1.12}$$

$$\vec{b} := \vec{d}_{p+l} \cup \vec{d'}_{p+l} \cup \vec{s}_0 \cup \vec{s'}_0 \cup \bigcup_{0 \le x \le p+l+r, x \ne (p+l)} (\vec{i}_x \cup \vec{i'}_x) \tag{1.13}$$

则 $\vec{a} \cup \vec{b}$ 包含了两个迁移函数展开序列上的所有输入状态向量 $< \vec{i}_0, \ldots, \vec{i}_{p+l+r} >$ and $< \vec{i'}_0, \ldots, \vec{i'}_{p+l+r} >$。他同时也包含了两个展开序列的初始状态 $\vec{s}_0$ 和 $\vec{s'}_0$。进一步的，等式 (1.11) 前两行的迁移关系 T 能够从输入序列和初始状态唯一的计算出输出序列。因此 $\vec{a}$ 和 $\vec{b}$ 能够唯一决定 $F'^d_{PC}(p,l,r,t)$ 中 t 的取值。因此，对于特定 p,l 和 r，以 $\vec{f}_{p+l}$ 为输入并使得 $F'^d_{PC}(p,l,r,1)$ 可满足的函数可以通过以 $F'^d_{PC}(p,l,r,t)$, $\vec{a}$ 和 $\vec{b}$ 为参数调用算法1.4得到：

$$FSAT_{PC}(p,l,r) := CharacterizingFormulaSAT(F'^d_{PC}(p,l,r,t), \vec{a}, \vec{b}, t) \tag{1.14}$$

因此 $FSAT_{PC}(p,l,r)$ 覆盖了使得 $F^d_{PC}(p,l,r)$ 可满足的 $\vec{f}_{p+l}$ 赋值集合。因此，其反 $\neg FSAT_{PC}(p,l,r)$ 是使得 $F^d_{PC}(p,l,r)$ 不可满足的 $\vec{f}_{p+l}$ 集合。

从命题1.1可知，$F^d_{PC}(p,l,r)$ 的不可满足证明可以推广到任意更大的 p,l 和 r 上。任意被 $\neg FSAT_{PC}(p,l,r)$ 覆盖的 $\vec{f}$ 也仍然能够使 $F^d_{PC}(p,l,r)$ 对于任意更大的 p,l 和 r 不可满足。

所以我们有：

命题 1.3: $\neg FSAT_{PC}(p,l,r)$ 是 $valid(\vec{f})$ 针对 p,l 和 r 单调递增的一个下估计。

这直观的展示在了图1.4中。

1.4.2.2 $valid(\vec{f})$

.

类似的，通过将公式 (1.2) 中 $F_{LN}(p,l,r)$ 的 i 替换为 $\vec{d}$，我们有：

$$F^d_{LN}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \ne \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{1.15}$$

如果 $F^d_{LN}(p,l,r)$ 可满足，则 $\vec{d}_{p+l}$ 不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。进一步的，类似于命题1.2，通过展开等式 (1.15) 中最后三行的环，我们能够证明 $\vec{d}_{p+l}$ 对于任意更大的 p,l 和 r 都不能被唯一决定。通过收集等式 (1.15) 的第三行和最后三行，我们进一步定义了 $T_{LN}(p,l,r)$:

$$T_{LN}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{1.16}$$

通过将等式 (1.15) 的第三行和最后三行替换为 $T_{LN}(p,l,r)$，我们得到:

$$F'^d_{LN}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{LN}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.17}$$

很显然 $F^d_{LN}(p,l,r)$ 和 $F'^d_{LN}(p,l,r,1)$ 等价。因此，对于特定的 p,l 和 r，定义在 $\vec{f}_{p+l}$ 上且能够使得 $F^d_{LN}(p,l,r)$ 可满足的函数可以通过下式计算:

$$FSAT_{LN}(p,l,r) := CharacterizingFormulaSAT(F'^d_{LN}(p,l,r,t),\vec{a},\vec{b},t) \tag{1.18}$$

再次参见命题1.2，$F^d_{LN}(p,l,r)$ 的可满足证明可以扩展到任意更大的 p,l 和 r 上。因此任意被 $FSAT_{LN}(p,l,r)$ 覆盖的 $\vec{f}$ 仍然能够对所有更大的 p,l 和 r 使得 $F^d_{LN}(p,l,r)$ 可满足。因此 $FSAT_{LN}(p,l,r)$ 单调增长且是 $\neg valid(\vec{f})$ 的子集。

因此我们下列命题:

命题 1.4: $\neg FSAT_{LN}(p,l,r)$ 是 $valid(\vec{(f)})$ 的单调递减上估计。

这被直观的展示在了图1.4中。

1.4.2.3 $valid(\vec{f})$

.

基于命题1.3 和1.4，我们给出了算法1.5以推导 $valid(\vec{f}_{p+l})$。该算法迭代的增加 p, l 和 r，直到 $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ 不可满足。这意味着

算法 1.5 *InferringUniqueFormula*: 推导使得 $\vec{d}_{p+l}$ 能够被唯一决定的 $valid(\vec{f}_{p+l})$

1: p:= p_{max}; l:= l_{max}; r:= r_{max} ;
2: **while** $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ is satisfiable **do**
3: p ++ ; l ++ ; r ++ ;
4: **end while**
5: **return** $\neg FSAT_{LN}(p,l,r)$

$FSAT_{PC}(p,l,r)$ 和 $FSAT_{LN}(p,l,r)$ 收敛到一个确定的集合上。在此情况下，$\neg FSAT_{PC}(p,l,r)$ 即为 $valid(\vec{f})$。

该算法的正确性证明在下一小节给出。

1.4.3 停机和正确性证明

首先我们需要证明以下三个引理:

引理 1.1: 算法1.5 中的 $\mathbf{FSAT_{PC}(p,l,r)}$ 针对 p, l 和 r 单调递减。

证明: 对于任意 $p' > p$,$l' > l$ 和 $r' > l$，假设 $A : \vec{f}_{p'+l'} \rightarrow B$ 是流控变量在第 $(p'+l')$ 步的取值。进一步假设 A 被 $FSAT_{PC}(p',l',r')$ 覆盖。

根据等式 (1.14) 算法1.4，易知 A 能够使得 $F'^{d}_{PC}(p',l',r',1)$ 可满足。假设 $F'^{d}_{PC}(p',l',r',1)$ 的满足赋值为 A'。易知 $A(\vec{f}_{p'+l'}) \equiv A'(\vec{f}_{p'+l'})$。

直观的，如图1.5所示，通过将 $(p'+l')$ 和 $(p+l)$ 步对准，我们能够将 $F'^{d}_{PC}(p',l',r',1)$ 的状态变量，输入变量和输出变量赋值映射到 $F'^{d}_{PC}(p,l,r,1)$。并淘汰前置和后置的状态序列。形式化的，对于 $p'+l'-l-p \leq n \leq p'+l'+r$，我们映射 $F'^{d}_{PC}(p',l',r',1)$ 中的 s_n 到 $F'^{d}_{PC}(p,l,r,1)$ 中的 $s_{n-p'-l'+l+p}$。i_n 和 o_n 的映射类似。

基于该映射，我们能将 $F'^{d}_{PC}(p',l',r',1)$ 的 A' 映射为另一个 $F'^{d}_{PC}(p,l,r,1)$ 的可满足赋值 A''。

通过将 A'' 的定义域限制为 $\vec{f}_{p+l}$，我们得到第四个可满足赋值 $A''' : \vec{f}_{p+l} \rightarrow B$。从上述构造过程可知，$A''' \equiv A$。

因此，任意被 $FSAT_{PC}(p',l',r')$ 覆盖的 A，都能够被 $FSAT_{PC}(p,l,r)$ 覆盖。

因此，$FSAT_{PC}(p,l,r)$ 针对 p, l 和 r 单调递减。 ■

引理 1.2: 算法1.5 中的 $\mathbf{FSAT_{LN}(p,l,r)}$ 针对 p, l 和 r 单调递增。

证明: 对于任意 $p' > p$,$l' > l$ 和 $r' > l$，假设 $A : \vec{f}_{p+l} \rightarrow B$ 是流控变量子在第 $(p+l)$ 步的赋值。进一步假设 A 被 $FSAT_{LN}(p,l,r)$ 覆盖。

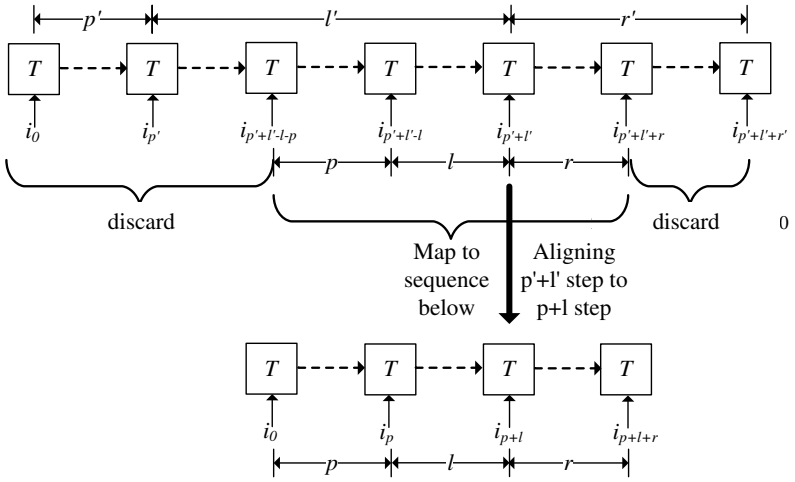


图 1.5　通过将第 $(p'+l')$ 步对准第 $(p+l)$ 步映射 $F'^{d}_{PC}(p',l',r',1)$ 到 $F'^{d}_{PC}(p,l,r,1)$。

因此 A 能够使 $F'^{d}_{LN}(p,l,r,1)$ 可满足。假设 $F'^{d}_{LN}(p,l,r,1)$ 的可满足赋值为 A'。可知 $A(\vec{f}_{p+l}) \equiv A'(\vec{f}_{p+l})$。

从图1.6可知，通过对齐第 $(p+l)$ 步到第 $(p'+l')$ 步，并展开三个环，我们能将 $F'^{d}_{LN}(p,l,r,1)$ 映射 $F'^{d}_{LN}(p',l',r',1)$。如此可得到 $F'^{d}_{LN}(p',l',r',1)$ 的可满足赋值 A''。

通过将 A'' 的定义域限制为 $\vec{f}_{p'+l'}$，我们可以得到第四个可满足赋值 A'''：$\vec{f}_{p'+l'} \to B$。很明显 $A \equiv A'''$。

这意味着每个被 $FSAT_{LN}(p,l,r)$ 覆盖的赋值也同时被 $FSAT_{LN}(p',l',r')$ 覆盖。因此 $FSAT_{LN}(p,l,r)$ 针对 p, l 和 r 单调递增。 ■

引理 1.3: $FSAT_{LN}(p,l,r) \Rightarrow FSAT_{PC}(p,l,r)$

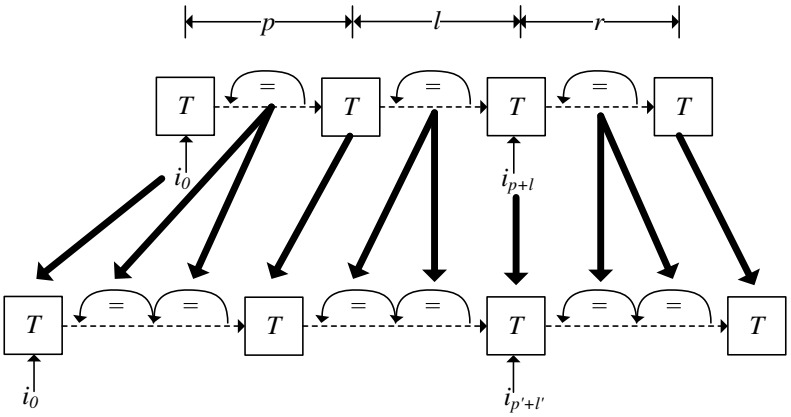


图 1.6　通过将第 $(p+l)$ 步对准 $(p'+l')$ 步并展开三个环，从而映射 $F'^{d}_{LN}(p,l,r,1)$ 到 $F'^{d}_{LN}(p',l',r',1)$。

证明：很明显 $F'^{d}_{LN}(p,l,r,1)$ 的短句集合是 $F'^{d}_{PC}(p,l,r,1)$ 的超集。所以 $F'^{d}_{LN}(p,l,r,1)$ 的每个可满足赋值也能够使得 $F'^{d}_{PC}(p,l,r,1)$ 可满足。因此 $FSAT_{LN}(p,l,r) \Rightarrow FSAT_{PC}(p,l,r)$ 成立。 ■

这三个引理直观的展示在图1.4中。很明显 $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ 在算法1.5中是单调递减的。基于这些引理，我们首先证明1.5 是停机的：

定理 1.1: 算法1.5 是停机算法。

证明：编码器是一个有限状态机，其最长的无环路径的长度是有限的。如果算法1.5 不停机，则 p, l 和 r 总会大于该长度。这意味着在下面的三个状态序列 $< \vec{s}_0,\ldots,\vec{s}_p >$,$< \vec{s}_{p+1},\ldots,\vec{s}_{p+l} >$ 和 $< \vec{s}_{p+l+1},\ldots,\vec{s}_{p+l+r} >$ 上必然存在环。因此，每个 $F'^{d}_{PC}(p,l,r,1)$ 的可满足赋值必然也能够满足 $F'^{d}_{LN}(p,l,r,1)$。这意味着 $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ 是不可满足的。这将导致算法1.5的停机。所以得证。 ■

其次，我们将证明算法1.5的正确性。

定理 1.2: 从算法1.5返回的 $\neg FSAT_{LN}(p,l,r)$ 覆盖且仅覆盖了所有能够使 $\vec{d}$ 被 $\vec{o}$ 的有限长度序列唯一决定的 $\vec{f}$。

证明：首先证明覆盖的情况。$FSAT_{LN}(p,l,r)$ 覆盖了一个 $\vec{f}$ 集合使得 $\vec{d}$ 对特定的 p, l 和 r 不能被唯一决定。因此 $\neg FSAT_{LN}(p,l,r)$ 排除了该集合，从而包含了所有能够使得 $\vec{d}$ 被唯一决定的 $\vec{f}$。

然后我们证明仅覆盖的情形。如果 A 是被 $\neg FSAT_{LN}(p,l,r)$ 覆盖的 $\vec{f}$ 的赋值，且能够使得 $\vec{d}$ 针对特定 p', l' 和 r' 不被唯一决定，则有：

1. 一方面 $FSAT_{LN}(p',l',r')$ 也覆盖 A。则从引理1.2可知，对所有 $p'' > max(p',p), l'' > max(l',l)$ 和 $r'' > max(r',r)$，$FSAT_{LN}(p'',l'',r'')$ 也覆盖 A。

2. 同时 $FSAT_{LN}(p,l,r)$ 不覆盖 A。$FSAT_{LN}(p,l,r)$ 是算法1.5 结束前计算出来的最后一个。这意味着 $valid(\vec{f})$ 的上估计和下估计收敛了。因此对于所有 $p'' > max(p',p), l'' > max(l',l)$ 和 $r'' > max(r',r)$，$FSAT_{LN}(p'',l'',r'')$ 必然等于 $FSAT_{LN}(p,l,r)$。因此 $FSAT_{LN}(p'',l'',r'')$ 也不覆盖 A。

这导致了冲突，因此仅覆盖的情形得证。 ■

算法 1.6 *RemoveRedundancy*(p, l, r)

for $r' := r \rightarrow 1$ **do**
  if $F_{PC}(p, l, r'-1) \wedge valid(\vec{f}_{p+l})$ 是可满足的 **then**
    break;
  else if $r' \equiv 1$ **then**
    $r' := r' - 1$;
    break ;
  end if
end for
for $l' := l \rightarrow 1$ **do**
  if $F_{PC}(p, l'-1, r') \wedge valid(\vec{f}_{p+l'-1})$ 是可满足的 **then**
    break;
  else if $l' \equiv 1$ **then**
    $l' := l' - 1$;
    break;
  end if
end for
return $< l', r' >$

1.5

我们首先在小节1.5.1中介绍为什么和如何削减 l 和 r 的长度。然后在小节1.5.2中给出我们算法的另一种可能结构。并讨论为什么我们选择了小节1.5.1 中的算法而不是小节1.5.2中的算法。

1.5.1 压缩 l 和 r

由于我们在算法1.5中同步增长 p,l 和 r 的值。这导致他们的值有可能包含冗余。这将导致产生的解码器的面积和时序上有不必要的额外开销。

例如，假设一个编码器仅包含一个简单的 buffer，其功能为 $o := i$。当 $p \equiv 0$,$l \equiv 0$ 和 $r \equiv 0$ 时，我们能够得到最简单的解码器 $i := o$。该解码器只包含一个 buffer，而不包含寄存器。而对于 $p \equiv 0$,$l \equiv 0$ 和 $r \equiv 1$，我们则需要一个额外的寄存器以将 o 延迟一步，然后从延迟的 o 中回复 i_i。

根据图1.2，很明显 r 影响解码器的电路面积和延迟，l 仅影响解码器的电路面积，而 p 并不对解码器的上述特性带来影响。

因此如算法1.6所示，我们选择首先压缩 r，然后压缩 l。

为了简化描述，我们将仅介绍 r 的情况。在行2，当 $F_{PC}(p,l,r'-1) \wedge valid(\vec{f}_{p+l})$ 可满足，则 r' 是最后一个使其不可满足的，我们将其直接返回 r'。另一方面，如果 $r' \equiv 0$ 仍能够使行4的 $F_{PC}(p,l,r') \wedge valid(\vec{f}_{p+l})$ 不可满足，我们直接返回 0。

1.5.2 另一种可能的算法结构

在上述讨论中，我们在算法1.3中同步增加 p, l 和 r 以找到流控变量，然后在算法1.6中压缩他们的值。该算法需要调用 SAT 求解器的次数是 $O(n)$ 。其中 $n = max(p,l,r)$。

还有另一种可能的方法：即使用三个嵌套的环逐一增加 p, l 和 r。该算法需要调用 SAT 求解器的次数是 $O(n^3)$ 。

我们将在小节1.7.6 中指出，同步增加 p, l 和 r 然后使用算法1.6 进行压缩比单独增加 p, l 和 r 要更有优势。我们将在那里对该现象进行解释。

1.6

在小节1.3中，解码器的输入 $\vec{i}$ 被划分为两个向量：流控向量 $\vec{f}$ 和数据向量 $\vec{d}$。为这两个向量分别产生解码器函数的算法是不同的，因此他们将在下面两个小节中分别描述。

1.6.1 产生 $\vec{f}$ 的解码器函数

每个 $f \in \vec{f}$ 都能够被输出向量 $\vec{o}$ 的有限长度序列唯一决定。所以，对于每个特定的输出向量序列 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$，$f_{p+l}$ 不能同时取值为 1 和 0。因此，计算 f_{p+l} 的解码器函数可以视为 ϕ_A 针对 ϕ_B 的 Craig 插值，其中 ϕ_A 和 ϕ_B 分别定义如下：

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & f_{p+l} \equiv 1 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \end{array} \right\} \tag{1.19}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & f'_{p+l} \equiv 0 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.20}$$

显然 $\phi_A \wedge \phi_B$ 等价于等式 (1.1) 中的 $F_{PC}(p,l,r)$ ，因此它是不可满足的。ϕ_A 和 ϕ_B 的共同变量集合为 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$。因此，Craig 插值 ITP 可以使用 McMillian

算法[9] 从 $\phi_A \wedge \phi_B$ 的不可满足证明序列中产生出来。ITP 覆盖了所有使得 $f_{p+l} \equiv 1$ 的 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 的赋值。同时，$ITP \wedge \phi_B$ 是不可满足的，因此 ITP 没有覆盖任何使 $f_{p+l} \equiv 0$ 成立的 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 的赋值。因此，ITP 是计算 $f \in \vec{f}$ 的解码函数。

为了进一步提高产生 $f \in \vec{f}$ 的解码函数的速度，我们通过以下方式使用 MiniSat 的递增求解机制：

1. 从 ϕ_A 中移除 $f_{p+l} \equiv 1$，从 ϕ_B 中移除 $f'_{p+l} \equiv 0$。
2. 将 $\phi_A \wedge \phi_B$ 加入 MiniSat 的短句数据库。
3. 针对每一个 $f \in \vec{f}$，使用 $f_{p+l} \equiv 1$ 和 $f'_{p+l} \equiv 0$ 作为求解的假设。并从不可满足证明中产生 Craig 插值。

1.6.2 产生 $\vec{d}$ 的解码函数

假设 $valid(\vec{f})$ 是被算法1.5推导出来的谓词。为每个 $d \in \vec{d}$，我们定义以下两个公式：

$$\phi'_A := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & d_{p+l} \equiv 1 \\ \wedge & valid(\vec{f}_{p+l}) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \end{array} \right\} \tag{1.21}$$

$$\phi'_B := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & d'_{p+l} \equiv 0 \\ \wedge & valid(\vec{f'}_{p+l}) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{1.22}$$

当 $valid(\vec{f})$ 成立时，每个 $d \in \vec{d}$ 均能够被唯一决定。因此，如果 $valid(\vec{f}_{p+l})$ 成立，对于每个特定的 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$，$d_{p+l}$ 不能同时为 1 和 0，因此，$\phi'_A \wedge \phi'_B$ 是不可满足的。因此，可以使用 McMillian 算法[9] 从 $\phi'_A \wedge \phi'_B$ 的不可满足证明中产生 Craig 插值 ITP。ITP 覆盖且仅覆盖使得 $d_{p+l} \equiv 1$ 的 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 的赋值。因此，ITP 是 $d \in \vec{d}$ 的解码器函数。

进一步的当 $valid(\vec{f}_{p+l})$ 不成立时，数据变量 $d \in \vec{d}_{p+l}$ 不能被唯一决定。因此，不存在计算它的解码器函数。不过这并不影响我们的算法的正确性，因为在这种情况下解码器只需恢复 $\vec{f}$，并忽略 $\vec{d}$。

类似的，我们也使用小节1.6.1 中的增量求解机制加速该算法。

1.7

我们使用 OCaml 语言实现了所有算法。并使用 MiniSat 1.14[15] 求解所有的 CNF 公式。所有的实验使用一台包含 16 个 Intel Xeon E5648 2.67GHz 处理器，192GB 存储器, 和 CentOS 5.4 Linux 操作系统的服务器进行。

1.7.1 测试集

表1.1 给出了测试集的信息。他们在自于两个来源：

1. 我们以前的论文[4].
2. Liu et al. [6].

表1.1 的每一列依次给出了每个实验对象的输入位数，输出位数，状态位数，映射到 mcnc.genlib 标准单元库后的门数和面积。映射使用 ABC 综合工具[19]，脚本为”strash; dsd; strash; dc2; dc2; dch; map”。该脚本来自于[6]。本文剩余部分给出的所有电路面积和延时都使用同样的设置产生。这使得我们的结果可以被用于和[6] 作比较。

表1.1 的最后一列也给出了我们将如何描述每一个 benchmark 的实验结果。

1. 对于来自于我们以前论文[4] 的 5 个 benchmark，我们发现他们中的大多数都有流控机制。这并不奇怪，因为这些 benchmark 都来自于实际的工业项目。他们的实验结果将分别在小节 1.7.2,1.7.3 和1.7.4中描述。
2. 对于其他不包含流控机制的 benchmark，如果他们的输入都能够被输出唯一决定，则我们的算法能够将他们所有的输入都识别成流控变量，并直接生成他们的解码器函数。他们的实验结果将在小节1.7.5中描述。

我们还进行了下列额外的实验：

在小节1.7.6中，我们将比较下列两种不同算法的时间开销：

1. 在算法1.5中同时增长 p, l 和 r，然后在算法1.6中压缩他们的值。
2. 在算法1.5中使用三个嵌套的循环分别增长 p, l 和 r。

在小节1.7.7中，我们将比较在算法1.6中是否压缩 p,l 和 r，导致的在算法运行时间，解码器面积和延时方面的区别。

最后在小节 1.7.8中，我们将比较我们算法产生的解码器和手工书写的解码器在电路面积和延迟方面的差别。

1.7.2 PCI Express 2.0 编码器

该编码器遵从 PCI Express 2.0 标准[11]。在删除了所有的注释和空行之后，其源代码包含 259 行 verilog。

表1.2给出了所有输入和输出的描述。根据 8b/10b 编码机制的描述[20], 当 $TXDATAK \equiv 0$ 时，$TXDATA$ 可以为任何值。而当 $TXDATAK \equiv 1$ 时，$TXDATA$ 只能是 1C, 3C, 5C, 7C, 9C, BC, DC, FC, F7, FB, FD 和 FE。因此，我们写了一个断言以剔除不在编码表内的情形，并将其嵌入迁移函数 T。

算法1.2 在 0.475 秒内识别出了流控向量 $\vec{f} := CNTL\_TXEnable\_P0$。而后算法1.5 在 1.22 秒内推导出了 $valid(\vec{f}) := CNTL\_TXEnable\_P0$。最后算法1.6 在 0.69 秒内得到压缩后的 $p := 4$, $l := 0$ 和 $r := 2$。最后产生解码器函数使用了 0.26 秒。最终的解码器包含 156 个门和 0 个寄存器，面积为 366，延迟为 7.6。

本文算法的创新之处在于其处理流控机制的能力。我们将展示器编码器如何将无效的数据向量映射到输出向量 $\vec{o}$。通过研究编码器的代码，我们发现，当且仅当 $CNTL\_TXEnable\_P0 \equiv 0$ 成立，也就是 $TXDATA$ 和 $TXDATAK$ 无效时，输

表 1.1 Benchmarks

	名字	个数 in/ouy	个数 reg/gate	电路 面积	编码器 描述	处理 方法
来自于[4]并有流控的测试集	PCIE2	10 / 11	22 / 149	326	PCIE 2.0 [11]	小节 1.7.2
	XGXS	10 / 10	17 / 249	572	10Gb 以太网 clause 48[10]	小节 1.7.3
	T2Eth	14 / 14	53 / 427	947	UltraSPARC T2 的以太网模块	小节 1.7.4
来自于[4]但没有流控的测试集	XFI	72 / 66	72 / 5086	12381	10Gb 以太网 clause 49[10]	在小节1.7.5 中比较我们的算法和 Liu[6]
	SCRAM-BLER	64/64	58/353	1034	增加数据中的 01 翻转	
来自于[6]的测试集	CC_3	1/3	6/22	54	长度为 3 的卷机码	
	CC_4	1/3	7/26	63	长度为 4 的卷机码	
	HM(7,4)	4/7	3/38	103	输入 4 输出 7 的汉明码	
	HM(15,11)	11/15	4/102	317	输入 11 输出 15 的汉明码	

出 $HSS\_TXELECIDLE$ 为 1. 因此，解码器将使用 $HSS\_TXELECIDLE$ 来唯一决定 $CNTL\_TXEnable\_P0$。

1.7.3 10G 以太网编码器 XGXS

该编码器 XGXS 遵从 IEEE 802.3 标准的[10] 的 clause 48 。在删除空行和注释后，其包含 214 行 verilog。

表1.3给出了输入和输出变量列表。该编码器也使用 8b/10b 编码机制[20] ，它包含两个输入：8 位的有待编码数据 *encode_data_in* ，和 1 位的控制字符标志位 *konstant* 。根据编码表[20]，当 $konstant \equiv 0$ 时，*encode_data_in* 可以是任何值。而当 $konstant \equiv 1$ 时，*encode_data_in* 只能是 1C, 3C, 5C, 7C, 9C, BC, DC, FC, F7, FB, FD 和 FE。因此，我们将一个手工给出的断言嵌入 T 以剔除不在编码表内的情形。

算法1.2 在 0.31 秒内识别了流控向量 $\vec{f}$:= *bad_code*。然后算法1.5 在 0.95 秒内推导了谓词 $valid(\vec{f}) := bad\_code$。再次算法1.6 在 0.52 秒内得到压缩结果 $p := 4$, $l := 0$ 和 $r := 1$。最后产生解码器使用了 0.17 秒。该解码器包含 163 个门和 0 个寄存器。面积为 370 ，延迟为 8.1。

虽然该算法使用了和上述 PCI Express 2.0 编码器相同的编码机制。然而它使用了完全不同的方法处理流控机制。该编码器并没有单独的输出用于表明输出的有效性。相反，所有输入的具体值及其有效性都统一编码在 *encode_data_out* 中。通过研究该编码器的源代码，我们发现当且仅当 $bad\_code \equiv 1$，既 *encode_data_in* 和 *konstant* 均无效时，输出变量 *encode_data_out* 将成为 0010111101。因此解码器能够使用 *encode_data_out* 来唯一决定 *bad_code*。

1.7.4 UltraSPARC T2 以太网编码器

这个编码器来自于 UltraSPARC T2 开源处理器。他遵从于 IEEE 802.3 标准[10] 的 clause 36。在删除空行和注释之后，它包含 864 行 verilog 源代码。

表 1.2 PCI Express 2.0 编码器的输入和输出变量描述

	变量名	宽度	描述
输入	$TXDATA$	8	有待编码的数据
	$TXDATAK$	1	1 意味着 $TXDATA$ 是一个控制字符, 0 意味着 $TXDATA$ 是普通数据
	$CNTL\_TXEnable\_P0$	1	1 意味着 $TXDATA$ 和 $TXDATAK$ 的有效性
输出	HSS_TXD	10	被编码的数据
	$HSS\_TXELECIDLE$	1	电磁空闲状态

表 1.3　10G 以太网编码器 XGXS 的输入和输出

	变量名	宽度	描述
输入	*encode_data_in*	8	将被编码的数据
	konstant	1	1 意味着 *encode_data_in* 是一个控制字符 0 意味着 *encode_data_in* 时普通数据
	bad_code	1	1 意味着 *konstant* 和 *encode_data_in* 是无效的
输出	*encode_data_out*	10	编码结果

表1.4给出了输入和输出变量的列表。该编码器同样使用 8b/10b 编码机制[20]，但是采用了另外一种流控机制，完全不同于上述两个编码器。有待编码的数据仍然是 8 位的 *txd*。然而并不存在单独的有效位，而是在一个 4 位的 *tx_enc_ctrl_sel* 中定义执行什么样的动作。细节如表1.5所示。很明显控制字符和流控机制被混合在 *tx_enc_ctrl_sel* 中。表1.5 的最后四种情形不能被唯一决定，因为他们无法和 ‘PCS_ENC_DATA 区分开来。因此我们使用一个断言将他们剔除。

算法1.2 在 3.76 秒内识别了流控信号 $\vec{f} := \{tx\_enc\_ctrl\_sel, tx\_en, tx\_er\}$。然后算法1.5 在 21.53 秒内推导了谓词 $valid(\vec{f}) := tx\_enc\_ctrl\_sel \equiv `PCS\_ENC\_DATA$。再次算法1.6 在 6.15 秒内得到压缩结果 $p := 5, l := 0$ 和 $r := 4$。最后产生解码器花费了 3.40 秒。解码器包含 401 个门和 9 个寄存器。面积为 920 ，延迟 10.2。

如表1.5的最后一列所示，前 5 种情况都有各自特殊的控制字符被赋予 *tx_10bdata*。因此解码器总能从 *tx_10bdata* 恢复出 *tx_enc_ctrl_sel*。

1.7.5　针对不具备流控机制的编码器比较我们的算法和现有算法

表1.6 针对不具备流控机制的编码器比较了我们的算法和[6] 的算法。

表 1.4　UltraSPARC T2 以太网编码器的输入输出列表

	变量名字	宽度	描述
输入	*txd*	8	有待编码的数据
	tx_enc_ctrl_sel	1	参见表1.5
	tx_en	1	传输使能
	tx_er	1	传输一个错误字符
输出	*tx_10bdata*	10	编码结果
	txd_eq_crs_ext	10	传输一个特殊错误字符 其中 $tx\_er \equiv 1$ 且 $txd \equiv 8'h0F$
	tx_er_d	1	传输一个错误字符
	tx_en_d	1	传输使能
	pos_disp_tx_p	1	正向 parity

表 1.5　UltraSPARC T2 以太网编码器的动作列表

动作名称	动作含义
‘PCS_ENC_K285	发送 K28.5 控制字符
‘PCS_ENC_SOP	发送 K27.7 控制字符
‘PCS_ENC_T_CHAR	发送 K29.7 控制字符
‘PCS_ENC_R_CHAR	发送 K23.7 控制字符
‘PCS_ENC_H_CHAR	发送 K30.7 控制字符
‘PCS_ENC_DATA	发送编码后的 txd
‘PCS_ENC_IDLE2	发送 K28.5 D16.2 序列
‘PCS_ENC_IDLE1	发送 D5.6 数据符号
‘PCS_ENC_LINK_CONFA	发送 K28.5 D21.5 序列
‘PCS_ENC_LINK_CONFB	发送 K28.5 D2.2 序列

对于从 XFI 到 HM(15,11) 的 6 个 benchmark，我们有他们的源代码，而[6] 给出了他们的实验结果。因此我们能在这里比较我们的算法和[6] 的结果。

通过比较第二和第三列之和和第六列，可见[6] 比本文算法快很多。尤其是第二列。这种差距的主要原因在于本文算法需要逐一检查每个 $i \in \vec{i}$ 是否能够被唯一决定，而[6] 可以在一次 SAT 求解之中检查所有的 $\vec{i}$。

另一个问题是本文算法的面积和延迟均大于[6]，原因在于本文算法所采用的 Craig 插值算法实现仍不够优化，可以通过移植 ABC [19] 的相应代码得到改善。

1.7.6　比较两种可能性：同时增长 p, l 和 r 或者单独增长

在算法1.3中，我们同时增加 p, l 和 r ，并在算法1.6中压缩他们的冗余值。我们称其为 A1 方案。

表 1.6　比较我们的算法和[6] 的算法

名字	我们的算法				[6]		
	检查解码器存在的时间开销	产生解码器的时间开销	解码器面积	解码器延迟	检查解码器存在和产生解码器的时间开销	解码器面积	解码器延迟
XFI	13.24	6.13	3878	13.8	8.59	3913	12.5
SCRAMBLER	1.80	0.55	698	3.8	0.42	640	3.8
CC_3	0.06	0.03	116	8.5	0.21	104	9.1
CC_4	0.16	0.09	365	12.5	0.20	129	9.0
HM(7,4)	0.09	0.03	258	8.1	0.05	255	7.3
HM(15,11)	1.49	2.23	5277	13.7	2.02	3279	13.2

表 1.7　比较两种可能性：同时增长 p, l 和 r 或者单独增长

benchmarks	A1: 同时增长					A2: 单独增长				
	p,l,r	时间识别 $\vec{f}$	时间推导 $valid(\vec{f})$	时间压缩 p,l,r	整体时间开下	p,l,r	时间识别 $\vec{f}$	时间推导 $valid(\vec{f})$	时间压缩 p,l,r	整体时间开销
PCIE2	3,0,2	0.49	1.21	0.68	2.38	3,0,2	0.38	0.80	0.38	1.60
XGXS	3,0,1	0.31	0.88	0.52	1.71	3,0,1	0.23	0.58	0.30	1.11
T2Eth	4,0,4	4.28	15.17	6.25	25.70	4,0,4	15.47	13.85	6.19	35.51
XFI	2,1,0	4.59	3.60	9.55	17.74	2,1,0	3.52	2.75	10.05	16.32
SCRAMBLER	2,1,0	0.64	0.58	1.33	2.55	2,1,0	0.48	0.43	1.47	2.38
CC_3	3,2,2	0.01	0.01	0.04	0.06	3,2,2	0.01	0.01	0.01	0.03
CC_4	4,4,3	0.07	0.01	0.08	0.16	4,1,4	0.16	0.01	0.07	0.25
HM(7,4)	3,0,0	0.02	0.01	0.07	0.09	3,0,0	0.01	0.01	0.04	0.06
HM(15,11)	3,0,0	0.22	0.05	1.21	1.49	3,0,0	0.34	0.04	0.58	0.96

小节1.5.2 给出了另一种可能性。它使用 3 个嵌套的循环来单独增长每一个 p, l 和 r 。我们称其为 À2 方案。

我们在表1.7中比较了这两种方案。

通过比较第 6 和 11 列中的整体时间开销，很显然 A2 在大多数情形下比 A1 快。只有 T2Eth 是一个例外。

这意味着我们应当使用 A2 而不是 A1 吗？答案是否定的。

从小节1.5.2可知，A1 需要调用 SAT 求解器的次数为 $O(n)$ ，其中 $n = max(p, l, r)$。而 A2 需要的次数为 $O(n^3)$ 。对于比较小的 n，两者并没有很大的区别。而对于较大的 n，比如 T2Eth，A1 对 A2 的优势是显著的。

因此 A2 在小电路上有优势，而 A1 在大电路上有优势。因此我们仍然选择 A1。也就是首先同步增加 p, l 和 r ，然后在算法1.6中压缩他们。

CC_4 是唯一一个在第 2 列和第 7 列具有不同 p,l 和 r 的 benchmark。这是由于 l 和 r 的不同增长顺序导致的。对于 A1 方案，其解码器包含 14 个寄存器，206 个门, 490 面积和 13.3 延迟。对于 A2 方案，其解码器包含 10 个寄存器, 61 个门, 154 面积和 9.6 延迟。因此 A2 方案比 A1 好很多。但是这仍然不意味着我们应当使用 A2。具体原因我们将在下一小节得到更多实验数据支撑之后进一步展开解释。

1.7.7　在压缩和不压缩 l 和 r 的两种算法之间比较运行时间，电路面积和延迟

为了改善解码器的面积和延迟，算法1.6 被用于在产生解码器之前压缩 l 和 r。表1.8展示了其效果。

表 1.8　在压缩和不压缩 *l* 和 *r* 的两种算法之间比较运行时间，电路面积和延迟

bench-marks	不压缩					使用算法1.6压缩					
	p,l,r	时间 生成 解码器	解码器 面积	寄存器 个数	最大 逻辑 延迟	时间 压缩 *p*,*l*,*r*	p,l,r	时间 生成 解码器	解码器 面积	寄存器 个数	最大 逻辑 延迟
PCIE2	3,3,3	0.44	382	11	7.5	0.68	3,0,2	0.28	366	0	7.6
XGXS	3,3,3	0.35	351	20	8.2	0.52	3,0,1	0.18	370	0	8.1
T2Eth	4,4,4	4.76	1178	9	10.9	6.25	4,0,4	3.41	920	9	10.2
XFI	2,2,2	10.67	5079	190	16.50	9.55	2,1,0	6.13	3878	58	13.8
SCRMBL	2,2,2	1.27	826	186	3.8	1.33	2,1,0	0.55	698	58	3.8
CC_3	3,3,3	0.04	117	11	9.2	0.04	3,2,2	0.03	116	9	8.5
CC_4	4,4,4	0.05	154	10	9.6	0.08	4,4,3	0.09	365	14	12.5
HM(7,4)	3,3,3	0.05	262	21	7.2	0.07	3,0,0	0.03	258	0	8.1
HM(15,11)	3,3,3	2.98	5611	45	13.5	1.21	3,0,0	2.23	5277	0	13.7

第一列是 benchmark 名字。当算法1.6 没有被使用时，第 2 列到第 6 列分别给出了 *p*,*l* 和 *r* 的值, 产生解码器的运行时间，解码器面积，解码器包含的寄存器个数，解码器最大的逻辑延迟。当算法1.6 被使用的时候，这些数据在最后 5 列给出。而第 7 列给出了 *l* 和 *r*。

通过比较 2-6 列和 8-12 列，很明显算法1.6显著的压缩了 *l* 和 *r* 。

CC_4 再次引起我们的注意。从第 4 列到第 6 列，我们发现电路面积和延迟非常类似于上一小节的 A2 情形。而他的 *p*,*l* 和 *r* 则类似于 A1 情形。这意味着分 CC_4 的解码器有至少两种差别很大的实现方式。而具体哪一种被选中取决于 SAT 求解器和 Craig 插值算法内部的某些不稳定因素。这回答了我们在上一小节的疑惑，即 A1 方案中的电路质量下降并不是由 A1 导致的。我们仍然应当使用 À1。

1.7.8　在我们的算法和手工书写的解码器之间比较电路面积和延迟

表1.9 在我们的算法和手工书写的解码器之间比较电路面积和延迟。CC_3,CC_4, HM(7,4) and HM(15,11) 没有在该表中是因为我们没有他们的手写解码器。

很明显在大多数情况下我们的解码器比手工解码器更小也更快。少数的两个例外是 T2Eth 和 XFI ，我们产生的解码器比手工解码器稍微大一些。

1.8

1.8.1 对偶综合

我们在[1] 中首次提出了对偶综合的概念。该算法通过迭代的增加迁移函数的展开长度以检查解码器是否存在，并通过传统的解遍历算法产生解码器函数。他的主要不足在于不停机，且产生解码器的时间开销太大。

我们[3] 和 Liu et al.[5] 独立的发现了如何通过检测环来得到停机的算法。而我们[4] 和 Liu et al.[5] 独立的发现可以通过 Craig 插值加速解码器的生成。

我们[4] 自动的发掘能够使得解码器存在的前提条件。该算法可视为本文算法1.5的特例。而本文算法1.5则是第一个允许在不同状态步上使用不同谓词的算法。

Tu et al.[7] 提出了一个突破性的算法，通过使用属性指导的可达性分析算法[21, 22]，将初始条件考虑到解码器存在性检测中。该算法是第一个能够考虑初始条件的对偶综合算法。

1.8.2 程序求反

文献[23] 指出，程序求反是指针对特定程序 P，求解其反程序 P^{-1}。因此，程序求反和我们的算法很类似。

程序求反的早期工作是基于证明的[24]，只能处理非常小的程序和非常简单的语法。

Glück et al. [25] 提出通过基于 LR 的分析方法消除非确定性，从而综合反程序。然而使用函数式语言使得该算法和我们的算法不兼容。

在文献[26] 中，Srivastava et al. 假设反程序和原始程序在结构上是相似的，共享相同的谓词集合和控制流结构。该算法通过在原始程序上迭代的推导和剔除非法路径，以获得合法的反程序。然而该算法不能保证完备性。

表 1.9　在我们的算法和手工书写的解码器之间比较电路面积和延迟

bench-marks	本文算法		手工书写的解码器	
	面积	最大逻辑演出	面积	最大逻辑延迟
PCIE2	366	7.6	594	9.7
XGXS	370	8.1	593	11.0
T2Eth	920	10.2	764	11.7
XFI	3878	13.8	3324	28.1
SCRAMBLER	698	3.8	1035	6.4

1.8.3 协议转换

协议转换是指在不同的通讯协议之间自动产生转换器。该领域和我们的工作是相关的，因为他们都试图自动产生通讯电路。

在文献[27, 28] 中，Avnit et al. 首先定义了一个通用的通讯协议模型，并给出了一个算法以检验是否存在某个协议的特定功能无法被翻译为另一个协议。最后他们给出了一个算法以计算目标协议的缓冲区控制函数的不动点。在文献[29] 中，他们引进了一个更高效的状态空间探索算法以提升整体性能。

1.8.4 可满足赋值遍历

绝大多数可满足赋值遍历算法致力于将一个完整的赋值扩展为一个包含较多赋值的赋值集合，以便减少调用 SAT 求解器的次数并压缩存储赋值解的空间开销。

文献[30] 提出了第一个此类算法。他在 SAT 求解器求解过程中构造一个蕴含图，用以记录每个赋值之间的依赖关系。每个不在该图中的赋值变量都可以从最终结果中剔除。在文献[31] 和[32] 中，每个变量如果在其不被约束的情况下不能使 $obj \equiv 0$ 被满足的话，则该变量可以从最终结果中剔除。在文献[33] 和[34, 35] 中，冲突分析方法被用于剔除与可满足性无关的变量。在文献[36] 中，变量集合被划分为重要变量和非重要变量集合。搜索过程中重要变量的优先级高于非重要变量。因此重要变量子集构成了一个搜索树，而该树的每一个叶节点是非重要变量的一个搜索子树。Cofactoring [16] 则通过将非重要变量设置为 SAT 求解器返回的值以缩减搜索空间。

另一类算法通过 Craig 插值以扩大解集合。文献[37] 提出了第一个此类算法。该算法构造两个相互矛盾的公式，并从他们的不可满足证明中抽取 Craig 插值。在文献[38] 中，Craig 插值的产生过程类似于传统的可满足赋值遍历算法。不过其扩展算法包含两步，分别对应于两个参与计算的公式。该算法是第一个不需要产生不可满足证明的 Craig 插值算法。

1.8.5 基于 Craig 插值的逻辑综合算法

在文献[39, 40] 中，函数依赖和逻辑分解问题被转换成一个两级布尔函数网络，其中基函数为第一级，而有待求解的函数为第二级，并用 Craig 插值算法产生。该算法也应用于我们的早期工作[4] 以找到所有可能的解码器。

在文献[41] 中，Craig 插值被用于产生 ECO。

在文献[37] 中，Craig 插值算法被用于从一个布尔关系中产生布尔函数。该算法也被应用于本文中以产生解码器。

1.9

本文提出了第一个能够处理流控机制的对偶综合算法。实验结果表明本文算法能够为多个来自于实际工业项目的复杂编码器，包括 PCI Express [11] 和以太网[10]。

trying to cite some papers

[42] [43]

[44]

[45]

参考文献

[1] Shen S, Zhang J, Qin Y, et al. Synthesizing complementary circuits automatically [C/OL]. In Proceedings of the 2009 International Conference on Computer-Aided Design. San Jose, CA, USA, 2009: 381–388. http://dx.doi.org/10.1145/1687399.1687472.

[2] Shen S, Qin Y, Wang K, et al. Synthesizing Complementary Circuits Automatically [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2010, 29 (8): 29:1191–29:1202. http://doi.acm.org/10.1109/TCAD.2010.2049152.

[3] Shen S, Qin Y, Xiao L, et al. A Halting Algorithm to Determine the Existence of the Decoder [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2011, 30 (10): 30:1556–30:1563. http://doi.acm.org/10.1109/TCAD.2011.2159792.

[4] Shen S, Qin Y, Wang K, et al. Inferring Assertion for Complementary Synthesis [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2012, 31 (8): 31:1288–31:1292. http://doi.acm.org/10.1109/TCAD.2012.2190735.

[5] Liu H-Y, Chou Y-C, Lin C-H, et al. Towards completely automatic decoder synthesis [C/OL]. In Proceedings of the 2011 International Conference on Computer-Aided Design, ICCAD 2011. San Jose, CA, USA, 2011: 389–395. http://dx.doi.org/10.1109/ICCAD.2011.6105359.

[6] Liu H-Y, Chou Y-C, Lin C-H, et al. Automatic Decoder Synthesis: Methods and Case Studies [J/OL]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2012, 31 (9): 31:1319–31:1331. http://doi.acm.org/10.1109/TCAD.2012.2191288.

[7] Tu K-H, Jiang J-H R. Synthesis of feedback decoders for initialized encoders [C/OL]. In Proceedings of the 50th Annual Design Automation Conference, DAC 2013. Austin, TX, USA, 2013: 1–6. http://dx.doi.org/10.1145/2463209.2488794.

[8] Abts D, Kim J. High Performance Datacenter Networks [M/OL]. 1st ed. Morgan and Claypool, 2011: 7–9. http://dx.doi.org/10.2200/S00341ED1V01Y201103CAC014.

[9] McMillan K L. Interpolation and sat-based model checking [M] // Warren A Hunt Jr F S. Computer Aided Verification, 15th International Conference, CAV 2003Vol.2725. Berlin Heidelberg: Springer-Verlag, 2003: 2003: 1–13.

[10] IEEE. IEEE Standard for Ethernet SECTION FOURTH. 2012. http://standards.ieee.org/getieee802/download/802.3-2012_section4.pdf.

[11] PCI-SIG. PCI Express Base 2.1 Specification. 2009. http://www.pcisig.com/members/downloads/specifications/pciexpress/PCI_Express_Base_r2_1_04Mar09.pdf.

[12] Moskewicz M W, Madigan C F, Zhao Y, et al. Chaff: Engineering an Efficient SAT Solver [C/OL]. In Proceedings of the 38th Design Automation Conference, DAC 2001. Las Vegas, NV, USA, 2001: 530–535. http://dx.doi.org/10.1145/378239.379017.

[13] Silva J P M, Sakallah K A. GRASP - a new search algorithm for satisfiability [C]. In Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design, ICCAD 1996. San Jose, CA, USA, 1996: 220–227.

[14] Goldberg E I, Novikov Y. BerkMin: a fast and robust SAT-solver [C]. In Proceedings of the conference on Design, automation and test in Europe, DATE 2002. Paris, France, 2002: 142–149.

[15] Eén N, Sörensson N. An extensible sat-solver [M] // Enrico Giunchiglia A T. Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003Vol.2919. Berlin Heidelberg: Springer-Verlag, 2003: 2003: 502–518.

[16] Ganai M K, Gupta A, Ashar P. Efficient sat-based unbounded symbolic model checking using circuit cofactoring [C/OL]. In Proceedings of the 2004 International Conference on Computer-Aided Design, ICCAD 2004. 2004: 510–517. http://dx.doi.org/10.1109/ICCAD.2004.1382631.

[17] Craig W. Linear reasoning: A new form of the herbrand-gentzen theorem [J]. The Journal of Symbolic Logic. 1957, 22 (3): 250–268.

[18] Zhang L, Madigan C F, Moskewicz M W, et al. Efficient Conflict Driven Learning in Boolean Satisfiability Solver [C]. In Proceedings of the 2001 International Conference on Computer-Aided Design, ICCAD 2001. 2001: 279–285.

[19] ABC:A system for sequential synthesis and verification. 2008. http://www.eecs.berkeley.edu/alanmi/abc/.

[20] Widmer A X, Franaszek P A. A DC-Balanced, Partitioned-Block, 8B/10B Transmission Code [J/OL]. IBM Journal of Research and Development. 1983, 27 (5): 440–451. http://dx.doi.org/10.1147/rd.275.0440.

[21] Bradley A R. SAT-based model checking without unrolling [M] // Ranjit Jhala D A S. Verification, Model Checking, and Abstract Interpretation, 12th International Conference, VMCAI 2011Vol.6538. Berlin Heidelberg: Springer-Verlag, 2011: 2011: 70–87.

[22] Eén N, Mishchenko A, Brayton R K. Efficient implementation of property-directed reachability [C]. In Proceedings of the International Conference on Formal Methods in Computer-Aided Design,FMCAD 2011. Austin, TX, USA, 2011: 125–134.

[23] Gulwani S. Dimensions in program synthesis [C/OL]. In Proceedings of the 12th international ACM SIGPLAN symposium on Principles and practice of declarative programming,PPDP 2010. Hagenberg, Austria, 2010: 13–24. http://dx.doi.org/10.1145/1836089.1836091.

[24] Dijkstra E W. Program Inversion [C]. In Proceeding of Program Construction, International Summer School. London, UK, 1979: 54–57.

[25] Glück R, Kawabe M. A method for automatic program inversion based on LR(0) parsing [J/OL]. Journal Fundamenta Informaticae. 2005, 66 (4): 367–395. http://doi.acm.org/10.1109/TCAD.2012.2191288.

[26] Srivastava S, Gulwani S, Chaudhuri S, et al. Path-based inductive synthesis for program inversion [C/OL]. In Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation,PLDI 2011. San Jose, CA, USA, 2011: 492–503. http://dx.doi.org/10.1145/1993498.1993557.

[27] Avnit K, Sowmya V D A, Parameswaran S R S. A Formal Approach To The Protocol Converter Problem [C/OL]. In Proceedings of the conference on Design, automation and test in Europe,DATE 2008. Munich, Germany, 2008: 294–299. http://dx.doi.org/10.1109/DATE.2008.4484695.

[28] Avnit K, D'Silva V, Sowmya A, et al. Provably correct on-chip communication: A formal approach to automatic protocol converter synthesis [J/OL]. ACM Transactions on Design Automation of Electronic Systems. 2009, 14 (2): 14:1–14:41. http://doi.acm.org/10.1145/1497561.1497562.

[29] Avnit K, Sowmya A. A formal approach to design space exploration of protocol converters [C/OL]. In Proceedings of the Conference on Design, Automation and Test in Europe,DATE 2009. 3001 Leuven, Belgium, Belgium, 2009: 129–134. http://dx.doi.org/10.1109/DATE.2009.5090645.

[30] McMillan K L. Applying sat methods in unbounded symbolic model checking [M] // Ed Brinksma K G L. International Conference on Computer Aided Verification, CAV 2002Vol.2404. Berlin Heidelberg: Springer-Verlag, 2002: 2002: 250–264.

[31] Ravi K, Somenzi F. Minimal assignments for bounded model checking [M] // Kurt Jensen A P. Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004Vol.2988. Berlin Heidelberg: Springer-Verlag, 2004: 2004: 31–45.

[32] Chauhan P, Clarke E M, Kroening D. A sat-based algorithm for reparameterization in symbolic simulation [C]. In Proceedings of the 41th Design Automation Conference, DAC 2004. 2004: 524–529.

[33] Shen S, Qin Y, Li S. Minimizing counterexample with unit core extraction and incremental sat [M] // Cousot R. Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005Vol.3385. Berlin Heidelberg: Springer-Verlag, 2005: 2005: 298–312.

[34] Jin H, Somenzi: F. Prime clauses for fast enumeration of satisfying assignments to boolean circuits [C/OL]. In Proceedings of the 42th Design Automation Conference, DAC 2005. 2005: 750–753. http://dx.doi.org/10.1109/DAC.2005.193911.

[35] Jin H, Han H, Somenzi F. Efficient conflict analysis for finding all satisfying assignments of a boolean circuit [M] // Nicolas Halbwachs L D Z. Tools and Algorithms for the Construction and Analysis of Systems, 11th International Conference, TACAS 2005Vol.3440. Berlin Heidelberg: Springer-Verlag, 2005: 2005: 287–300.

[36] Grumberg O, Schuster A, Yadgar A. Memory efficient all-solutions sat solver and its application for reachability analysis [M] // Alan J Hu A K M. International Conference on Formal Methods in Computer-Aided Design,FMCAD 2011Vol.3312. Berlin Heidelberg: Springer-Verlag, 2004: 2004: 275–289.

[37] Jie-Hong Roland Jiang W-L H, Hsuan-Po Lin. Interpolating functions from large Boolean relations [C]. In Proceedings of 2009 International Conference on Computer-Aided Design. 2009: 779–784.

[38] Chockler H, Ivrii A, Matsliah A. Computing Interpolants without Proofs [M] // Armin Biere T E J V, Amir Nahir. 8th International Haifa Verification Conference, HVC 2012Vol.7857. Berlin Heidelberg: Springer-Verlag, 2012: 2012: 72–85.

[39] Lee C-C, Jiang J-H R, Huang C-Y, et al. Scalable exploration of functional dependency by interpolation and incremental SAT solving [C]. In Proceedings of 2007 International Conference on Computer-Aided Design. 2007: 227–233.

[40] Lee R-R, Jiang J-H R, Hung W-L. Bi-decomposing large Boolean functions via interpolation and satisfiability solving [C/OL]. In Proceedings of the 45th Design Automation Conference, DAC 2008. 2008: 636–641. http://dx.doi.org/10.1145/1391469.1391634.

[41] Wu B-H, Yanga C-J, Huang C-Y, et al. A robust functional ECO engine by SAT proof minimization and interpolation techniques [C/OL]. In Proceedings of 2010 International Conference on Computer-Aided Design. 2010: 729–734. http://dx.doi.org/10.1109/ICCAD.2010.5654265.

[42] Barthel W, Hartmann A K, Leone M, et al. Hiding solutions in random satisfiability problems: A statistical mechanics approach [J/OL]. CoRR. 2001, cond-mat/0111153. http://arxiv.org/abs/cond-mat/0111153.

[43] Qin Y, Shen S, Kong J, et al. Cloud-Oriented SAT Solver Based on Obfuscating CNF Formula [C/OL] // Han W, Huang Z, Hu C, et al. In Web Technologies and Applications - APWeb 2014 Workshops, SNA, NIS, and IoTS, Changsha, China, September 5, 2014. Proceedings. 2014: 188–199. http://dx.doi.org/10.1007/978-3-319-11119-3_18.

[44] Yongzhi Wang, Jinpeng Wei, Mudhakar Srivatsaywa. CROSS CLOUD MAPREDUCE: A RESULT INTEGRITY CHECK FRAMEWORK ON HYBRID CLOUDS. to appear in International Journal of Cloud Computing (ISSN 2326-7550).

[45] Benjamin D, Atallah M J. Private and Cheating-Free Outsourcing of Algebraic Computations [C/OL] // Korba L, Marsh S, Safavi-Naini R. In Sixth Annual Conference on Privacy, Security and Trust, PST 2008, October 1-3, 2008, Fredericton, New Brunswick, Canada. 2008: 240–245. http://dx.doi.org/10.1109/PST.2008.12.