分类号 TP393 学号 09069030

UDC 密级 公开

工学博士学位论文

基于白盒模型的对偶综合

博士生姓名 秦莹

学科专业 计算机科学与技术

研究方向 计算机软件与理论

指导教师 贾焰 研究员

国防科学技术大学研究生院

二〇一五年八月

Complementary Synthesis based on White Box Model

Candidate: Qin Ying

Supervisor: Professor Jia Yan

A dissertation

Submitted in partial fulfillment of the requirements

for the degree of Doctor of Engineering

in Computer Science and Technology

Graduate School of National University of Defense Technology

Changsha, Hunan, P. R. China

August 15, 2015

独 创 性 声 明

本人声明所呈交的学位论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表和撰写过的研究成果，也不包含为获得国防科学技术大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文题目：基于白盒模型的对偶综合

学位论文作者签名：_____ 日期： 年 月 日

学位论文版权使用授权书

本人完全了解国防科学技术大学有关保留、使用学位论文的规定。本人授权国防科学技术大学可以保留并向国家有关部门或机构送交论文的复印件和电子文档，允许论文被查阅和借阅;可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密学位论文在解密后适用本授权书。）

学位论文题目：基于白盒模型的对偶综合

学位论文作者签名：_____ 日期： 年 月 日

作者指导教师签名：_____ 日期： 年 月 日

目 录

表 目 录

图 目 录

摘　要

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 [1–8] 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。

而另一方面，现代复杂通讯协议的编码器中，广泛采用了流控 [9] 和流水线等复杂内部结构，以提升编码器的性能和对复杂应用环境的适应性。而目前在对偶综合方面的所有研究工作[1–8] 均基于黑盒模型，从而无法发挥上述内部结构在性能和适应性方面的优势。

为了克服上述问题，本文基于白盒模型，探索了如何在对偶综合中发掘编码器的内部结构信息，如流控和流水线结构，以自动产生支持相应结构的解码器。本文的主要研究内容及创新点包括以下几方面:

第一，研究了基于余因子 (Cofactoring) 和 Craig 插值[10] 的迭代特征化算法。在发掘编码器内部结构和自动产生解码器的过程中，一个必须而且对性能要求非常苛刻的步骤，是特征化满足特定命题逻辑关系 R 的布尔函数 f。传统的算法包括基于 SAT 或 BDD 的完全解遍历和和量词削减。然而这些算法通常受到解空间不规则的困扰，导致性能低下。为此，我们创造性的提出了一个迭代的特征化算法框架。在每一次迭代中，为每一个尚未被遍历的解 A，利用其对应的余因子化简 R 以满足产生 Craig 插值要求。而该插值是 A 的一个充分扩展。该迭代过程是停机的，且其性能比传统的完全解遍历算法有巨大的提升。

第二，研究了针对流控机制的对偶综合算法。传统对偶综合算法的[1–8] 的一个基本假设是，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。基于该假设方可构造满足 Craig 插值的不可满足公式。然而，许多高速通讯系统的编码器所带有流控机制[9]，直接违反了上述假设。该机制将 $\vec{i}$ 划分为有待编码的数据向量 $\vec{d}$ 和用以表达 $\vec{d}$ 有效性的流控向量 $\vec{f}$，并在 $\vec{f}$ 上定义一个有效性谓词 $valid(\vec{f})$。只有在 $valid(\vec{f}) \equiv 1$ 的情形下，$\vec{d}$ 才能够被 $\vec{o}$ 唯一决定。为此，我们创造性的提出了能够处理流控机制的对偶综合算法：**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[11] 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 $valid(\vec{f}) \equiv 1$ 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 $valid(\vec{f}) \equiv 1$。

第三，研究了针对流水线结构的对偶综合算法。现代集成电路中的编码器，为了提升工作频率，通常包含多个流水线级，以将关键的数据路径划分为多级。而传统的对偶综合算法[1–8] 完全无视这种流水线结构，从而导致生成的解码器无法保持和编码器匹配的频率和性能。为此，我们创造性的提出了能够产生流水解码器的对偶综合算法：首先将传统对偶综合算法推广到非输入输出情形，以找到编码器中每一个流水线级 $s\vec{t}g^{j}$ 中的寄存器集合；然后使用迭代 Craig 插值算法特征化每一个流水线级 $s\vec{t}g^{j}$ 的布尔函数，以从下一个流水线级 $s\vec{t}g^{j+1}$ 或输出 $\vec{o}$ 之中恢复 $s\vec{t}g^{j}$。最终特征化 $\vec{i}$ 的布尔函数以从第一个流水线级 $s\vec{t}g^{0}$ 中恢复 $\vec{i}$。

第四，结合上述研究成果，研究了能够同时处理流控和流水线结构的对偶综合算法。该算法首先使用秦 et al. [12] 的算法来寻找 $\vec{f}$ 并推导 $valid(\vec{f})$。然后分别通过强制和不强制 $valid(\vec{f})$，已从所有寄存器集合中找到每一个寄存器级 $s\vec{t}g^{j}$ 的 $\vec{d}^{j}$ 和 $\vec{f}^{j}$。最后通过 Jiang et al. [13] 的算法特征化 $s\vec{t}g^{j}$ 和 $\vec{i}$ 的布尔函数。

综上所述，本文对基于白盒模型的对偶综合算法中若干关键问题进行了深入的研究，提出了针对流控和流水线结构的解决方案。理论分析和实验结果验证了所提出算法的有效性和性能，对于进一步促进对偶综合算法的发展和应用具有一定的理论意义和应用价值。

关键词: **对偶综合；流控；流水线；余因子；**Craig **插值**

ABSTRACT

One of the most difficult jobs in designing communication and multimedia chips is to design and verify complex encoder and decoder pairs. The encoder maps its input variables $\vec{i}$ to its output variables $\vec{o}$, while the decoder recovers $\vec{i}$ from $\vec{o}$. Complementary synthesis [1–8] eases this job by automatically generating a decoder from an encoder, with the assumption that $\vec{i}$ can always be uniquely determined by a bounded sequence of $\vec{o}$.

On the other hand, The encoders of modern communication protocols widely employ flow control[9] and pipeline mechanism to boost performance and their ability of adopting to complex application environment. But state-of-the-art research works on complementary synthesis[1–8] are all based on black box model, so can not take full advantage of these mechanisms.

To overcome these problems, based on white box model, this paper propose new complementary synthesis algorithms to handle flow control and pipeline mechanism:

First, this thesis proposes an iterative characterizing algorithm based on cofactoring and Craig interpolant. In inferring the encoder's internal structure and generating its decoder, one of the most critical algorithm is to characterize a Boolean function f that covers a Boolean relation R. State-of-the-art algorithms are satisfying assignment enumeration and quantifier elimination with SAT or BDD. But these algorithms are very inefficient because of irregular solution space. Thus we propose an iterative characterizing algorithm. In each iteration of this algorithm, for each satisfying assignment not yet enumerated A, R is simplified with A's cofactor, and then is used to generated a Craig interpolant, which can be used as an enlarged cube of A. This algorithm is a halting one and is much faster than naive satisfying assignment enumeration.

Second, this thesis proposes a novel complementary synthesis algorithm to handle flow control mechanism. One assumption of State-of-the-art complementary algorithms[1–8] is that, $\vec{i}$can always be uniquely determined by a bounded sequence of $\vec{o}$. Only in this way the unsatisfiable formula for computing Craig interpolant can be constructed. But the flow control mechanism [9] in many modern encoders fail this assumption. This mechanism partition $\vec{i}$ into the data vector $\vec{d}$ to be encoded and the flow control vector $\vec{f}$ used to express the validness of $\vec{d}$. And a validness predicate $valid(\vec{f})$ is defined on $\vec{f}$. $\vec{d}$ can be uniquely determined by $\vec{o}$ only when $valid(\vec{f}) \equiv true$. Thus this thesis propose

a novel complementary synthesis algorithm to handle flow control mechanism: First, it identifies all input variables that can be uniquely determined, and takes them as flow control variables. Second, it infers a predicate over these flow control variables that enables all other input variables to be uniquely determined. Third, it characterizes the decoder's Boolean function with Craig interpolant.

Third, this thesis proposes a novel complementary synthesis algorithm to handle pipeline structure. Modern encoders often include multiple pipeline stages that cut the critical datapath into multiple segments to boost frequency and thus performance, State-of-the-art complementary algorithms [1–8] all ignore such structure, which make the generated decoder can not keep up with the encoder on frequency and performance. Thus this thesis proposes a novel algorithm to first find out the encoder's pipeline registers in each pipeline stage, and then characterize all Boolean functions that recover each of these pipeline registers from the registers in the next pipeline stage, and finally characterize the Boolean functions that recover the encoder's input variables from the first pipeline stage.

Finally, with all above researches, this thesis proposes a final algorithm to handle flow control mechanism and pipeline structure at the same time. First, it infers the flow control predicate on inputs with Qin et al. [12]'s algorithm. Second, it finds out the pipeline stages in the encoder by enforcing the inferred flow control predicate. Finally, the decoder's Boolean functions that recover each pipeline stage and input are characterized with Jiang et al. [13]'s algorithm based on Craig interpolant.

This thesis proposes several algorithm to exploit the encoder's internal structure in complementary synthesis. Theoretical analysis and experimental result indicate that these algorithm are useful and efficient.

Key Words: Complementary synthesis; Flow control mechanism; Pipeline structure; Cofactoring ; Craig interpolant

第一章　绪论

通讯和多媒体应用是半导体工业的主要推动力。这两个领域日新月异的发展，带来了对传输带宽永无止境的追求。常见的高性能传输协议，如以太网、InfiniBand 和 PCI Express 等，其单通道带宽从本世纪初的 3.125Gbps 增长到目前的 25Gbps。

为了克服在高频信号衰减方面的挑战，每一代新的传输标准都会采用全新的编码方案。因此，在通讯和多媒体芯片设计项目中，最为关键且困难的工作之一是设计和验证特定的物理层编码器和解码器。针对这一关键而困难的工作，工业界常见的设计方法仍然停留在简单地手工编写代码，并使用动态模拟器进行验证的阶段。

而随着传输带宽的进一步提升，对于编码后信号白平衡性能的要求也日益严格 (加一个注释)。这就导致了在编码器中引进了更为复杂的，包含大量异或操作的线性移位寄存器组，如以太网标准 clause 49 的 64/66 加扰器和 PCI Express 标准的 128/130 编码器。大量的异或操作使得编码器的映射表非常不规则，包含的 corner case 的数量随编码长度指数增长。这就对传统的动态模拟验证方法的完备性提出了严峻挑战。

为此，国防科技大学的沈胜宇于 2009 年首次提出了对偶综合的概念和基本的算法实现[1]，以从一个通讯协议的编码器源代码中，自动产生其对应的解码器代码。以此为起点，集成电路设计自动化领域的研究者们在该领域取得了大量的研究成果[1–8]。

我们将在本章的剩余部分中首先介绍相关的背景知识，然后详细描述目前在该领域的研究成果。

1.1　背景知识

本节将给出相关的背景知识。

1.1.1　基本记法

布尔集合记为 $B = \{0, 1\}$。多个变量组成的向量记为 $\vec{v} = (v, \dots)$。$\vec{v}$ 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。

1.1.2 命题逻辑可满足性问题

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[14]，Grasp[15]，Berkmin[16]，和 MiniSat[17]。

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[18] 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[10]。ϕ_I 可以使用 McMillan 算法[11] 得到。Craig 插值通常被用于产生 ϕ_A 的上估计。

1.1.3 MiniSat 求解器的递增求解机制

本文中，我们使用 MiniSat 求解器[17] 求解所有 CNF 公式。和其他基于冲突学习机制[19] 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。

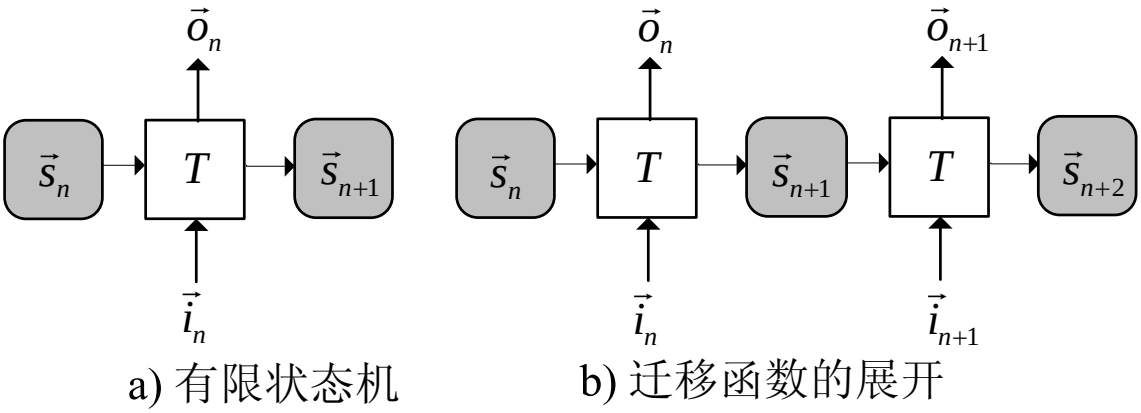


图 1.1　有限状态机及其迁移关系的展开

1.1.4　有限状态机

如图1.1a) 所示，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 $\vec{o}$。

如图1.1b) 所示，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　对偶综合研究现状

本节将简述到目前为止，对偶综合领域取得的研究成果。由于其中的若干算法也将在本论文的主要章节中被使用，因此其描述方式根据本论文的需要进行了修改。因此有可能和原始论文的描述方式有轻微出入。在阅读本论文是应以本小节的描述为准。

1.2.1　早期的充分非完备算法

国防科技大学的沈胜宇[1] 首次提出了对偶综合的概念和初步算法实现。该算法是充分非完备的，这意味着当特定编码器对应的解码器存在时，该算法总能找到其实现；而当解码器不存在时，该算法不停机。

对于每一个 $i \in \vec{i}$，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 (5.1) 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 (5.1) 的行 1 对应于图5.3左边的路径，而行 2 对应于图5.3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5.3可知，等式 (5.1) 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。

从图5.3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 (5.1) 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。

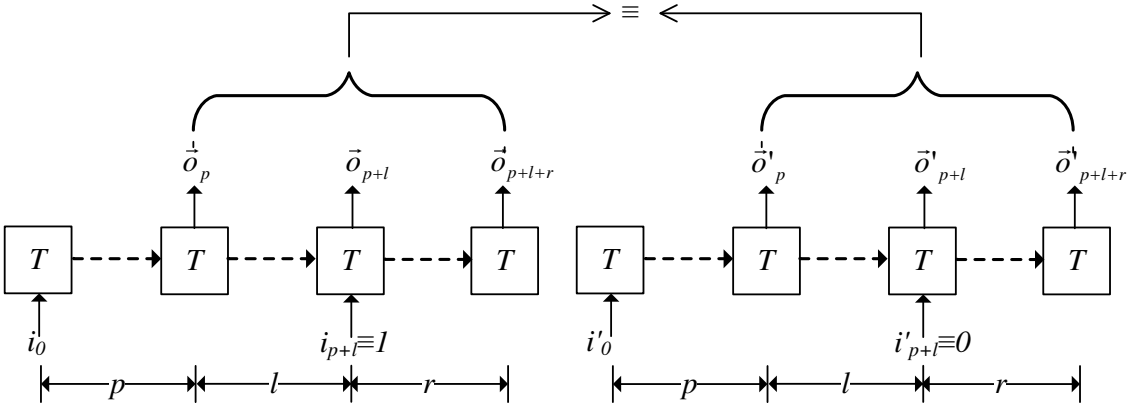


图 1.2　用于检查 i_{p+l} 是否能够被唯一决定的下估计算法

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。

上述命题仅考虑了输入向量 $\vec{i}$ 中的一个输入变量 i。当需要讨论整个 $\vec{i}$ 时，我们需要将整体的 p, l 和 r 设置为最大值。即，假设对于 $i \in \vec{i}$，其对应的唯一决定参数为 p_i, l_i 和 r_i，则对于整个 $\vec{i}$，其对应的唯一决定参数为:

$$\begin{aligned} p &:= max_{i\in\vec{i}}\{p_i\} \\ l &:= max_{i\in\vec{i}}\{l_i\} \\ r &:= max_{i\in\vec{i}}\{r_i\} \end{aligned} \tag{1.2}$$

等式 (5.1) 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[8] 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。

1.2.2 完备停机算法

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。国防科技大学的沈胜宇[3] 和 Liu et al.[6] 分别独立提出了类似的全新的解决方案。该方案如图3.3所示，该图类似于图5.3，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 (5.2) 中。

$$F_{LN}(p,l,r) := \left\{ \begin{array}{ll} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{1.3}$$

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。

根据命题4.1 和4.2, 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4.1，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4.1，可知 $F_{PC}(p',l',r')$

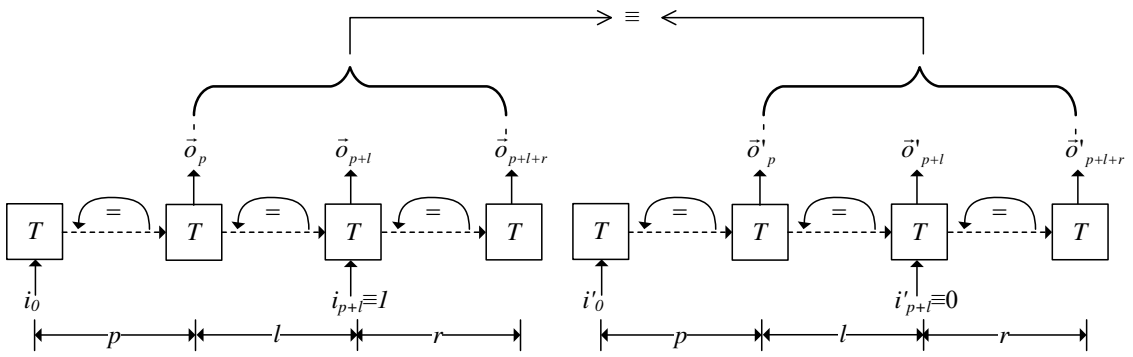


图 1.3　用于检查 i_{p+l} 是否不能被唯一决定的上估计算法

算法 1.1 *CheckUniqueness*(*i*): 用于检测 $i \in \vec{I}$ 是否能够被 $\vec{\sigma}$ 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
```

是不可满足的。因此 $F_{PC}(p,l,r)$ 总能够在算法4.1行5成为不可满足的并退出循环;

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7被满足。这样将导致退出循环。

因此该算法是停机的。

1.2.3 在对偶综合领域的其他的相关工作

沈胜宇[5] 和 Liu et al.[6] 独立的发现可以通过 Craig 插值加速解码器的生成。该算法还将作为下一章的一部分单独介绍，因此不在这里展开描述。

沈胜宇[5] 自动的发掘能够使得解码器存在的前提条件。并首次发现在同一个前提条件下，存在多个不同的解码器。为了方便用户区分和选择，该文还提出了单独的算法来寻找所有这些解码器，并推导他们存在的前提条件。

Tu et al.[8] 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[20, 21]，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。

1.3 基于白盒模型的对偶综合

本节阐述基于白盒模型的对偶综合技术的重要研究意义，并讨论从事该研究所面临的主要挑战。

1.3.1 研究意义

现代复杂通讯协议的编码器中，广泛采用了流水线和流控机制 [9] 技术。以提高性能并增强对环境的适应能力。而目前在对偶综合方面的所有研究工作[1–8]均基于黑盒模型，完全忽视上述的内部结构，从而无法发挥上述内部结构在性能和适应性方面的优势。

1.4 研究内容与创新点

为了克服上述问题，本文基于白盒模型，探索了如何在对偶综合中发掘编码器的内部结构信息，如流控和流水线结构，以自动产生支持相应结构的解码器。本文的主要研究内容及创新点包括以下几方面：

本文受国家自然科学基金项目“面向通讯应用的自动对偶综合方法研究”（项目编号 61070132）的支持，主要贡献和创新点如下：

第一，研究了基于余因子 (Cofactoring) 和 Craig 插值[10] 的迭代特征化算法。在发掘编码器内部结构和自动产生解码器的过程中，一个必须而且对性能要求非常苛刻的步骤，是特征化满足特定命题逻辑关系 R 的布尔函数 f。传统的算法包括基于 SAT 或 BDD 的完全解遍历和和量词削减。然而这些算法通常受到解空间不规则的困扰，导致性能低下。为此，我们创造性的提出了一个迭代的特征化算法框架。在每一次迭代中，为每一个尚未被遍历的解 A，利用其对应的余因子化简 R 以满足产生 Craig 插值要求。而该插值是 A 的一个充分扩展。该迭代过程是停机的，且其性能比传统的完全解遍历算法有巨大的提升。

第二，研究了针对流控机制的对偶综合算法。传统对偶综合算法的[1–8] 的一个基本假设是，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。基于该假设方可构造满足 Craig 插值的不可满足公式。然而，许多高速通讯系统的编码器所带有流控机制[9]，直接违反了上述假设。该机制将 $\vec{i}$ 划分为有待编码的数据向量 $\vec{d}$ 和用以表达 $\vec{d}$ 有效性的流控向量 $\vec{f}$，并在 $\vec{f}$ 上定义一个有效性谓词 $valid(\vec{f})$。只有在 $valid(\vec{f}) \equiv 1$ 的情形下，$\vec{d}$ 才能够被 $\vec{o}$ 唯一决定。为此，我们创造性的提出了能够处理流控机制的对偶综合算法：**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[11] 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 $valid(\vec{f}) \equiv 1$ 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 $valid(\vec{f}) \equiv 1$。

第三，研究了针对流水线结构的对偶综合算法。现代集成电路中的编码器，为了提升工作频率，通常包含多个流水线级，以将关键的数据路径划分为多级。而传统的对偶综合算法[1–8] 完全无视这种流水线结构，从而导致生成的解码器无法保持和编码器匹配的频率和性能。为此，我们创造性的提出了能够产生流水解码器的对偶综合算法：首先将传统对偶综合算法推广到非输入输出情形，以找到编码器中每一个流水线级 $s\vec{t}g^{j}$ 中的寄存器集合；然后使用迭代 Craig 插值算法特征化每一个流水线级 $s\vec{t}g^{j}$ 的布尔函数，以从下一个流水线级 $s\vec{t}g^{j+1}$ 或输出 $\vec{o}$ 之中恢复 $s\vec{t}g^{j}$。最终特征化 $\vec{i}$ 的布尔函数以从第一个流水线级 $s\vec{t}g^{0}$ 中恢复 $\vec{i}$。

第四，结合上述研究成果，研究了能够同时处理流控和流水线结构的对偶综合算法。该算法首先使用秦 et al. [12] 的算法来寻找 $\vec{f}$ 并推导 $valid(\vec{f})$。然后分别通过强制和不强制 $valid(\vec{f})$，已从所有寄存器集合中找到每一个寄存器级 $s\vec{t}g^{j}$ 的 $\vec{d}^{j}$ 和 $\vec{f}^{j}$。最后通过 Jiang et al. [13] 的算法特征化 $s\vec{t}g^{j}$ 和 $\vec{i}$ 的布尔函数。

综上所述，本文对基于白盒模型的对偶综合算法中若干关键问题进行了深入的研究，提出了针对流控和流水线结构的解决方案。理论分析和实验结果验证了所提出算法的有效性和性能，对于进一步促进对偶综合算法的发展和应用具有一定的理论意义和应用价值。

1.5 论文组织结构

论文共分七章，组织结构如下：

第一章为绪论，介绍了相关的背景知识，对偶综合的基本概念、特点、应用以及研究现状。分析基于白盒模型的对偶综合技术的研究意义，并简述本文的研究内容和组织结构。

第二章描述了基于余因子和 Craig 插值[10] 的迭代特征化算法。该算法在推导控制流谓词和特征化解码器的布尔函数中被广泛使用。

第三章描述了面向流控机制的对偶综合算法。

第四章描述了面向流水线结构的对偶综合算法。

第五章将上述两个算法有机的结合在一起，能够处理同时包含流控和流水线结构的编码器。

第六章总结全文并展望未来的工作。

最后是致谢、博士期间撰写的论文、参加的科研工作以及参考文献。

第二章　基于余因子和 Craig 插值的迭代特征化算法

2.1　问题描述

在形式化验证和综合领域，对于两个存在某种内在联系的逻辑向量 $\vec{a}$ 和 $\vec{b}$，有两种不同的方式表达他们之间的联系：关系和函数。

其中，关系 $R(\vec{a},\vec{b})$ 更具一般性，能够表达 $\vec{a}$ 和 $\vec{b}$ 之间的任意对应。尤其是一对多的对应，这是关系比函数具有更强描述能力的地方。这种一般性在形式化验证中广泛用于描述非确定性行为以扩展描述能力，以及构造抽象模型[22] 以削减计算复杂性等。

而另一方面，函数是关系的一种受限形式。如果 $R(\vec{a},\vec{b})$ 满足以下要求，则能将其转换为相应的函数 $\vec{b} := f(\vec{a})$：对 $\vec{a}$ 的任意取值 $x \in [\![\vec{a}]\!]$，均存在且仅存在唯一的 $y \in [\![\vec{b}]\!]$，使得 $R(\vec{a},\vec{b})$。在实际的软硬件设计与验证领域，存在大量的情况需要从一个关系中获得相应的函数。如在自动激励生成算法中从约束描述产生相应的激励函数[23]，证明导引抽象中的抽象模型构造[24]，以及本文中推导控制流谓词和特征化解码器等。

以图2.1为例。对于图2.1a) 中的一对一映射，我们可以使用布尔函数 $y_1 = x_1 \wedge x_2$ 和 $y_2 = \neg x_1 \wedge \neg x_2$ 表示。而另一方面，对于图2.1b) 中的布尔关系，并不存在相应的布尔函数，因为 $(x_1, x_2) = (0, 1)$ 的情形被映射到了多个 (y_1, y_2) 的组合。而这种情况可以使用布尔关系 $R = (\neg x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge y_2) \vee (\neg x_1 \wedge x_2 \wedge \neg y_1 \wedge \neg y_2) \vee (\neg x_1 \wedge x_2 \wedge y_1 \wedge y_2) \vee (x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge \neg y_2) \vee (x_1 \wedge x_2 \wedge y_1 \wedge \neg y_2)$。

在本章中，我们首先在小节2.2描述 Craig 插值的基本原理，然后在小节2.4中将其扩展到包含额外变量 $\vec{c}$ 的更一般情形 $R(\vec{a},\vec{b},\vec{c})$。

2.2　Craig 插值的原理和实现

注意本小节描述的 Craig 插值基本原理不是我们的原创，而是为了方便上下文的叙述从[25] 移植至此。

2.2.1　相关背景知识和记法

在通常的 SAT 求解器，包括本文使用的 MiniSat[17] 中，要求待求解的公式被表示为 CNF 格式。其中一个公式是多个短句的合取 (conjunction)，而每一个短句是多个文字的析取 (disjunction)，而每个文字是一个布尔变量 v 或者其反 $\neg v$。如公式 $(v_0 \vee \neg v_1 \vee v_2) \wedge (v_1 \vee v_2) \wedge (\neg v_0 \vee v_2)$，包含短句 $v_0 \vee \neg v_1 \vee v_2$,$v_1 \vee v_2$ 和 $\neg v_0 \vee v_2$。而短句 $v_0 \vee \neg v_1 \vee v_2$ 包含文字 v_0, $\neg v_1$ 和 v_2。

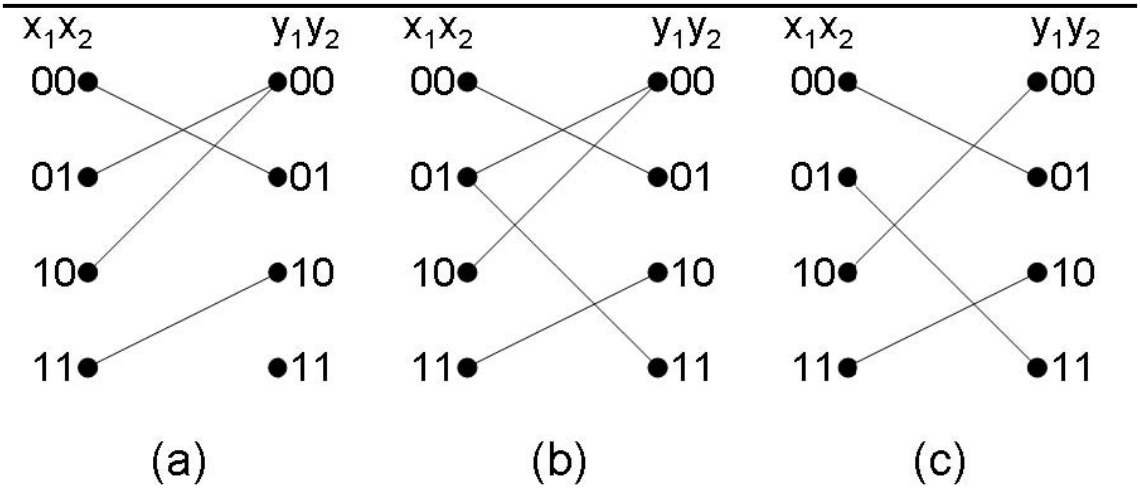


图 2.1　关系和函数的布尔映射

当存在一个变量 v，使得一个短句 c 中同时包含两个文字 v 和 $\neg v$，则称 c 为 tautological 的。我们通常假设有待 SAT 求解器求解的公式中所有的短句都是非 tautological 的。

当存在一个假设公式 F 的布尔变量全集为 V。若存在对 V 的赋值函数 $A : V \to \{0,1\}$，使得 F 中的每个短句均能取值为 1，则称 F 是可满足的，且 SAT 求解器能够找到赋值函数 A。否则称 F 为不可满足的。

2.2.2　不可满足证明

对于两个短句 $c_1 = v \vee B$ 和 $c_2 = \neg v \vee C$，当 $A \vee B$ 是非 tautological 的时，$A \vee B$ 称为他们的 resolvant。而 v 称为他们的 pivot。易知以下事实:

$$\begin{aligned} resolvant(c_1, c_2) &= \exists v, c_1 \wedge c_2 \\ c_1 \wedge c_2 &\to resolvant(c_1, c_2) \end{aligned} \tag{2.1}$$

定义 2.1: 对于不可满足公式 F，假设其短句集合为 C，则其不可满足证明 Π 是一个有向无环图 (V_Π, E_Π)，其中 V_Π 是短句集合，满足如下要求：

1. 对于节点 $c \in V_\Pi$：

   (a) 要么 $c \in C$，此时称 c 为 Π 的根

   (b) 或者 c 有且仅有两个 predecessors c_1 和 c_2，使得 c 是 c_1 和 c_2 的 resolvant。

2. 空短句是 Π 的唯一一个叶节点。

直观的说，Π 就是一棵树，以短句集合 C 的子集为根，以空短句为唯一叶节点。而每个节点的两个扇入边代表了一个 resolving 关系。

包括本文使用的 MiniSat 求解器[17] 在内的许多 SAT 求解器，当公式不可满足时都将产生一个不可满足证明 Π。

2.2.3 Craig 插值算法

根据文献[10]，给定两个布尔逻辑公式 A 和 B，若 $A \wedge B$ 不可满足，则存在仅使用了 A 和 B 共同变量的公式 I，使得 $A \Rightarrow I$ 且 $I \wedge B$ 不可满足。I 被称为 A 针对 B 的 Craig 插值[10]。

目前最常见且最高效的产生 Craig 插值的算法是 McMillan 算法[11]。其基本原理描述如下。

对于上述公式 A 和 B，已知 $A \cup B$ 不可满足，而 Π 是 SAT 求解器给出的不可满足证明。当一个变量 v 同时出现在 A 和 B 中时，我们称其为全局变量。若 v 只出现在 A 中，则称其为 A 本地变量。

对于文字 v 或者 $\neg v$，当变量 v 是全局变量或者 A 本地变量时，称该文字为全局文字或者 A 本地文字。

对于短句 c，令 $g(c)$ 为 c 中所有全局文字的析取，而 $l(c)$ 为 c 中所有 A 本地文字的析取。

例如，假设有两个短句 $c_1 = (a \vee b \vee \neg c)$ 和 $c_2 = (b \vee c \vee \neg d)$。并假设 $A = \{c_1\}$ 和 $B = \{c_2\}$。则 $g(c_1) = (b \vee \neg c)$，$l(c_1) = (a)$，$g(c_2) = (b \vee c)$，$l(c_2) = FALSE$。

定义 2.2: 令 (A, B) 为一对公式，而 Π 是 $A \cup B$ 的不可满足证明，且其叶节点是 $FALSE$。对于每一个节点 $c \in V_\Pi$，令 $p(c)$ 为如下定义的一个公式：

1. 如果 c 是根节点则

   (a) 如果 $c \in A$ 则 $p(c) = g(c)$

   (b) 否则 $p(c) = TRUE$

2. 否则令 c_1 和 c_2 分别是 c 的两个扇入节点，而 v 是他们的 pivot 变量

   (a) 如果 v 是 A 本地变量，则 $p(c) = p(c_1) \vee p(c_2)$。

   (b) 否则 $p(c) = p(c_1) \wedge p(c_2)$。

上述定义2.2是构造性的，已经给出了从 Π 得到最终的插值的算法，即以 Π 的根节点为起点，为每一个 c 计算相应的 $p(c)$，直至到达最终的唯一叶节点 $FALSE$。我们有以下定理：

定理 2.1: 定义2.2为唯一叶节点 $FALSE$ 产生的 $p(FALSE)$ 即为 A 相对于 B 的 Craig 插值。

该定理的详细证明可见文献[26]。

计算 A 相对于 B 的 Craig 插值的时间复杂性为 $O(N+L)$，其中 N 是 Π 中包含的节点个数 $|V_\Pi|$，而 L 是 Π 中的文字个数 $\sigma_{c\in V_\Pi}|c|$。而所产生的插值可以视为一个电路，其空间复杂性为 $|O(N+L)|$。当然，Π 的尺寸在最坏情况下也是 $A\cup B$ 的尺寸的指数。

2.3 非迭代的特征化算法

假设有布尔关系 $R(t,\vec{a})$ 使得 $R(1,\vec{a})\wedge R(0,\vec{a})$ 不可满足。而我们需要从 R 中特征化函数 f，使得 $t=f(\vec{a})$。则根据上述的讨论，可以简单的令 $A=R(1,\vec{a})$ 而 $B=R(0,\vec{a})$。此时 A 相对于 B 的 Craig 插值，即为函数 f 的一个实现。

该算法在本文的后继章节中被广泛应用于构造解码器的布尔函数。

2.4 迭代的特征化算法

上一小节讨论了如何从关系 $R(a,\vec{b})$ 特征化函数 $a=f(\vec{b})$。然而在更一般的情形下，我们需要从 $R(\vec{a},\vec{b},t)$ 特征化函数 $t=f(\vec{a})$。相比之下，此时多了一个需要进行存在性量化的 $\vec{b}$。为此我们需要将上述算法进行以下扩展。

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：

$$FSAT_R(\vec{a}) := \begin{cases} 1 & \exists\vec{b}.R(\vec{a},\vec{b},1) \\ 0 & otherwise \end{cases} \tag{2.2}$$

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 [18] 和 Craig 插值[11]，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[18] 可以构造以下公式：

算法 2.1 *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** $FSAT_R(\vec{a})$

$$R(\vec{a}, A(\vec{b}), 1) := R(\vec{a}, \vec{b}, 1)_{b \equiv A(b)} \tag{2.3}$$

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，$ITP(\vec{a}) \wedge R(\vec{a}, A(\vec{b}), 0)$ 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5.2 以特征化等式 (3.7) 中的 $FSAT_R(\vec{a})$。行3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5 和6 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7 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[11] 计算 Craig 插值 $ITP(\vec{a})$。

$ITP(\vec{a})$ 将在行8被加入 $FSAT_R(\vec{a})$ 并在行3 被排除。

算法5.2 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5.2 是停机的。

2.5 本章小结

本章综述了 Craig 插值算法的原理及其实现，以及基于该实现的迭代是特征化算法。这些算法将在本文的剩余部分被其他算法频繁调用。

第三章　面向流控机制的对偶综合

3.1　引言

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 [1–3, 5–8] 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[9]，而该功能直接违反了上述假设。图3.1a) 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3.1b) 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。

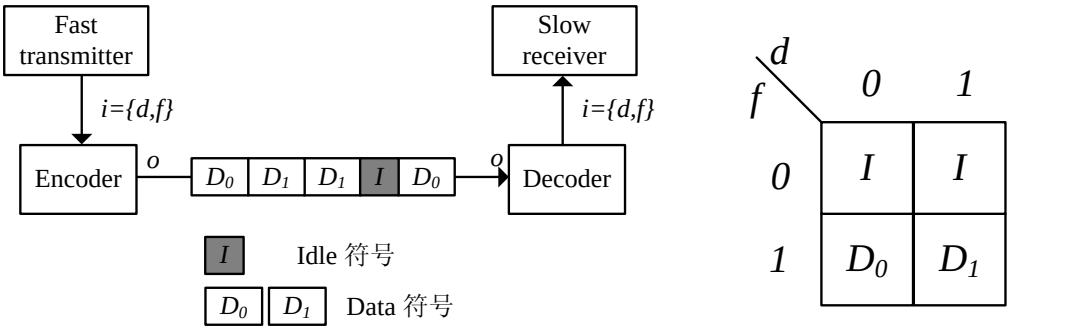
流控机制的工作原理为：

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3.1b) 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 [1–3, 5–8] 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：First, it identifies all input variables that can be uniquely determined, and takes them as flow control variables. Second, it infers a predicate over these flow control variables that enables all other input variables to be uniquely determined. Third, it characterizes the decoder's Boolean function with Craig interpolant.



a) 快速transmitter和慢速的receiver被编码器和解码器链接起来

b) 编码器的映射表将 d和 f 映射到o

图 3.1　带有流控机制的通讯系统及其编码器

该算法的第二步类似于[5]，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 [5] 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[5] 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 [27] 和 PCI Express [28])，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5.2节介绍了背景知识；第3.3节给出了识别流控变量的算法，而第3.4节推导使得 $\vec{d}$ 能够被 $\vec{o}$ 唯一决定的谓词；第3.5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5.5节给出了特征化解码器电路的算法；第5.6和5.7节分别给出了实验结果和相关工作。最后，第5.8节给出了结论。

3.2　背景知识

本节介绍相关的背景知识。

3.2.1　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 $\vec{v} = (v, \dots)$。$\vec{v}$ 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 $A : V \to B$，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[14]，Grasp[15]，Berkmin[16]，和 MiniSat[17]。

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[18] 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[10]。ϕ_I 可以使用 McMillan 算法[11] 得到。Craig 插值通常被用于产生 ϕ_A 的上估计。

3.2.2 MiniSat 求解器的递增求解机制

本文中，我们使用 MiniSat 求解器[17] 求解所有 CNF 公式。和其他基于冲突学习机制[19] 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。

3.2.3 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 $n \leq j < m$ 均满足。一个环是一条路径 $< \vec{s}_n, \ldots, \vec{s}_m >$ 使得 $\vec{s}_n \equiv \vec{s}_m$。

3.2.4 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 [1–3, 5–8] 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 [1–3, 5–8] 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5.2.3.1节中描述的下估计算法。而第二个估计算法是第5.2.3.2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4.2.3.3 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。

3.2.4.1 下估计算法

如图5.3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 (5.1) 中的公式 $F_{PC}(p, l, r)$ 的不可满足。

$$F_{PC}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.1}$$

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 (5.1) 的行 1 对应于

图5.3左边的路径，而行 2 对应于图5.3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5.3可知，等式 (5.1) 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。

从图5.3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 (5.1) 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 3.1: 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 (5.1) 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[8] 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

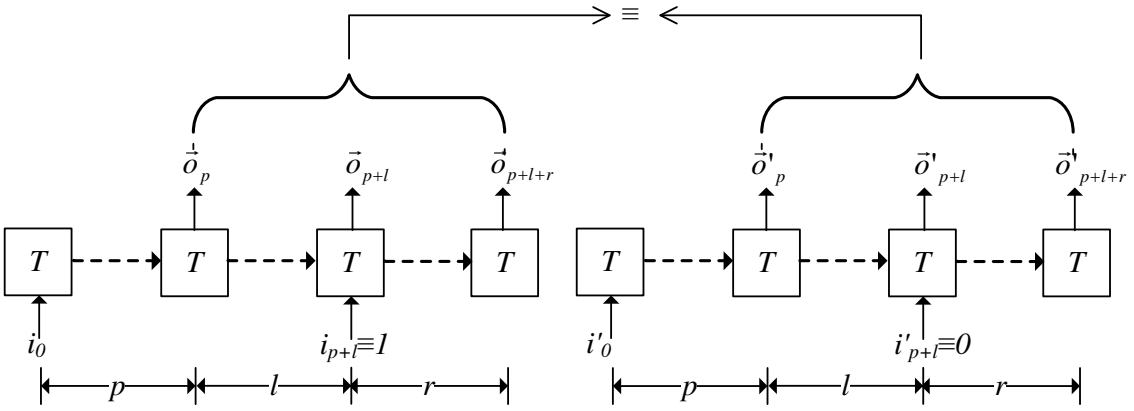


图 3.2　用于检查 i_{p+l} 是否能够被唯一决定的下估计算法

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。

3.2.4.2 上估计算法

.

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3.3所示，该图类似于图5.3，但是增加了三个约束用于检测三个路径

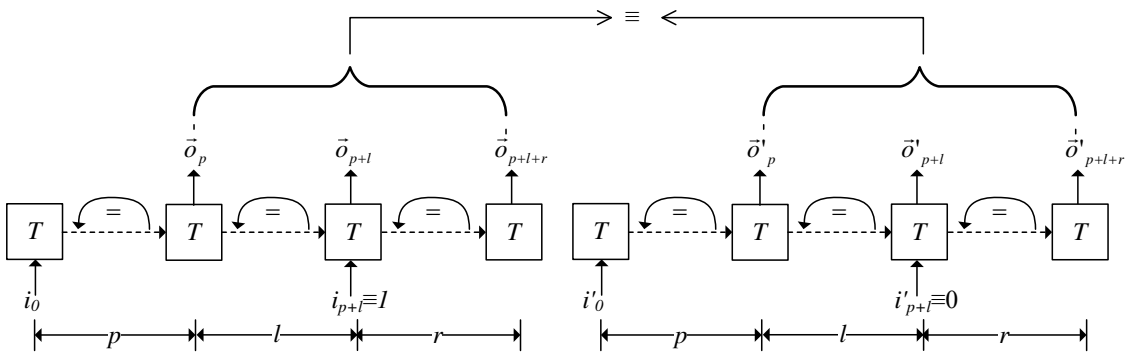


图 3.3 用于检查 i_{p+l} 是否不能被唯一决定的上估计算法

算法 3.1 *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
```

$< \vec{s}_0, \ldots, \vec{s}_p >$,$< \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ 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 (5.2) 中。

$$F_{LN}(p,l,r) := \left\{ \begin{array}{ll} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{3.2}$$

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 3.2: *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。*

3.2.4.3 完整算法

.

根据命题4.1 和4.2, 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4.1，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4.1，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4.1行5成为不可满足的并退出循环;

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7被满足。这样将导致退出循环。

因此该算法是停机的。

3.3 识别流控变量

本节将介绍如何识别流控变量 $\vec{f}$。这将首先在小节3.3.1中介绍。然后小节3.3.2将介绍如何使用增量求解算法加速该算法。

3.3.1 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。

算法 3.2 *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);
```

所以我们提出算法5.1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7，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5.1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5.5中被正确特征化。

3.3.2 使用增量 SAT 求解加速识别算法

通过节3.2.2中的 MiniSat 增量求解机制，我们能进一步加速算法5.1 。

通过将等式 (5.1) 的第三行移动到等式 (3.4)，等式 (5.1) 中的 $F_{PC}(p,l,r)$ 可以被划分为以下两个等式：

$$C_{PC}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.3}$$

$$A_{PC}(p,l,r) := \left\{ \; i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \; \right\} \tag{3.4}$$

类似的我们可以将等式 (5.2) 中的 $F_{LN}(p,l,r)$ 划分为下列两个等式：

$$C_{LN}(p,l,r) := \left\{ \begin{array}{ll} & C_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{3.5}$$

$$A_{LN}(p,l,r) := \left\{ \; i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \; \right\} \tag{3.6}$$

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5.1 修改成为算法3.3 。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3.2.2中描述的 MiniSat 的增量求解机制接口。

3.4 推导使得数据变量向量被唯一决定的谓词

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3.4.1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

算法 3.3 $FindFlowIncSAT(\vec{i})$: 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)
```

算法 3.4 *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

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

法。在小节5.2.4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。

3.4.1 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:

$$FSAT_R(\vec{a}) := \begin{cases} 1 & \exists \vec{b}.R(\vec{a},\vec{b},1) \\ 0 & otherwise \end{cases} \tag{3.7}$$

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 [18] 和 Craig 插值[11]，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[18] 可以构造以下公式:

$$R(\vec{a}, A(\vec{b}), 1) := R(\vec{a},\vec{b},1)_{b \equiv A(b)} \tag{3.8}$$

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，$ITP(\vec{a}) \wedge$

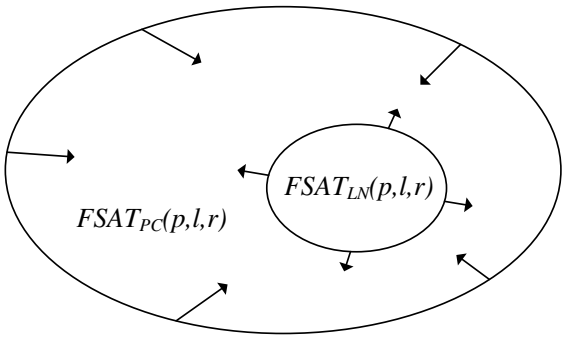


图 3.4　$FSAT_{PC}(p,l,r)$ 和 $FSAT_{LN}(p,l,r)$ 的单调性

$R(\vec{a},A(\vec{b}),0)$ 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5.2 以特征化等式 (3.7) 中的 $FSAT_R(\vec{a})$。行3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5 和6 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7 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[11] 计算 Craig 插值 $ITP(\vec{a})$。

$ITP(\vec{a})$ 将在行8被加入 $FSAT_R(\vec{a})$ 并在行3 被排除。

算法5.2 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5.2 是停机的。

3.4.2　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5.4所示，我们将首先在小节5.2.4.2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5.2.4.3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5.2.4.4中我们指出这两个估计将收敛至 $valid(\vec{f})$。小节3.4.3将证明该算法的正确性。

3.4.2.1 **计算 $valid(\vec{f})$ 的单调增长的下估计**

通过将等式 (5.1) 中的 i 替换为 $\vec{d}$，我们得到：

$$F^{d}_{PC}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.9}$$

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 (5.3) 的第三行，我们得到 $T_{PC}(p,l,r)$：

$$T_{PC}(p,l,r) := \left\{ \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \right\} \tag{3.10}$$

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：

$$F'^{d}_{PC}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{PC}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.11}$$

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：

$$\vec{a} := \vec{f}_{p+l} \tag{3.12}$$

$$\vec{b} := \vec{d}_{p+l} \cup \vec{d'}_{p+l} \cup \vec{s}_0 \cup \vec{s'}_0 \cup \bigcup_{0\leq x\leq p+l+r, x\neq(p+l)} (\vec{i}_x \cup \vec{i'}_x) \tag{3.13}$$

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 (5.5) 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5.2得到：

$$FSAT_{PC}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{PC}(p,l,r,t),\vec{a},\vec{b},t) \tag{3.14}$$

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4.1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 3.3: $\neg FSAT_{PC}(p,l,r)$ 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5.4中。

3.4.2.2 计算 $valid(\vec{f})$ 的单调递减上估计

.

类似的，通过将公式 (5.2) 中 $F_{LN}(p,l,r)$ 的 i 替换为 $\vec{d}$，我们有：

$$F^{d}_{LN}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r}\vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{3.15}$$

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4.2，通过展开等式 (3.15) 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 (3.15) 的第三行和最后三行，我们进一步定义了 $T_{LN}(p,l,r)$:

$$T_{LN}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{3.16}$$

通过将等式 (3.15) 的第三行和最后三行替换为 $T_{LN}(p,l,r)$，我们得到:

$$F'^{d}_{LN}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & t \equiv T_{LN}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.17}$$

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:

$$FSAT_{LN}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{LN}(p,l,r,t), \vec{a}, \vec{b}, t) \tag{3.18}$$

再次参见命题4.2，$F^{d}_{LN}(p,l,r)$ 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 3.4: $\neg FSAT_{LN}(p,l,r)$ 是 $valid(\vec{(f)})$ 的单调递减上估计。

这被直观的展示在了图5.4中。

3.4.2.3 计算 $valid(\vec{f})$ 的算法

.

基于命题3.3 和3.4，我们给出了算法5.3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 $FSAT_{PC}(p,l,r)$ 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。

算法 3.5 *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** $\neg FSAT_{LN}(p,l,r)$

3.4.3 停机和正确性证明

首先我们需要证明以下三个引理:

引理 3.1: 算法5.3 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 (5.8) 算法5.2, 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3.5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 3.2: 算法5.3 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。

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3.6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''。

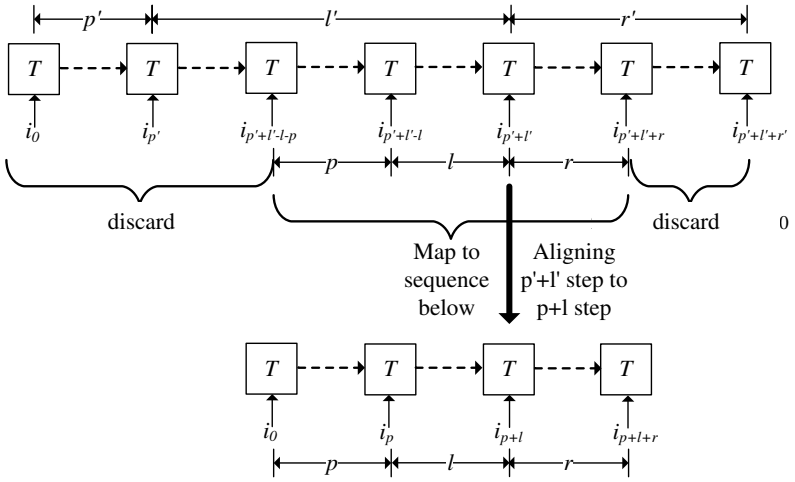


图 3.5　通过将第 $(p'+l')$ 步对准第 $(p+l)$ 步映射 $F'^{d}_{PC}(p',l',r',1)$ 到 $F'^{d}_{PC}(p,l,r,1)$。

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'''：$\vec{f}_{p'+l'} \rightarrow B$。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 3.3: $FSAT_{LN}(p,l,r) \Rightarrow FSAT_{PC}(p,l,r)$

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5.4中。很明显 $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ 在算法5.3中是单调递减的。基于这些引理，我们首先证明5.3 是停机的：

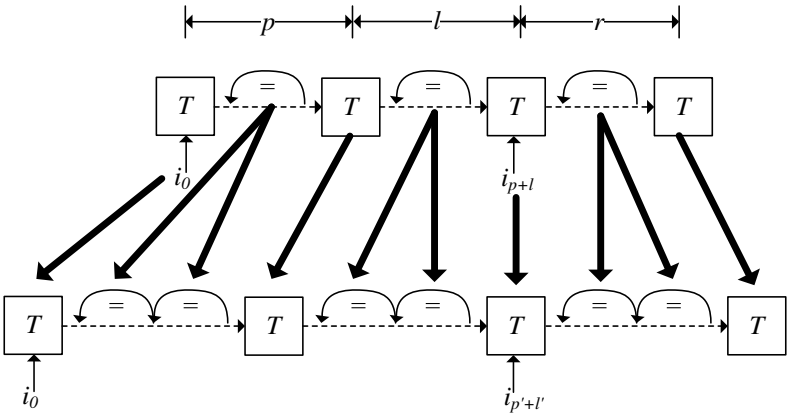


图 3.6　通过将第 $(p+l)$ 步对准 $(p'+l')$ 步并展开三个环，从而映射 $F'^{d}_{LN}(p,l,r,1)$ 到 $F'^{d}_{LN}(p',l',r',1)$。

定理 3.1: 算法5.3 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5.3 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5.3的停机。所以得证。 ■

其次，我们将证明算法5.3的正确性。

定理 3.2: 从算法5.3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3.2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5.3 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。 ■

3.5 压缩迁移关系展开序列的长度

我们首先在小节5.4.1中介绍为什么和如何削减 l 和 r 的长度。然后在小节3.5.2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5.4.1 中的算法而不是小节3.5.2中的算法。

算法 3.6 *RemoveRedundancy*(p, l, r)

1: **for** $r' := r \rightarrow 1$ **do**
2: **if** $F_{PC}(p, l, r'-1) \wedge valid(\vec{f}_{p+l})$ 是可满足的 **then**
3: break;
4: **else if** $r' \equiv 1$ **then**
5: $r' := r' - 1$;
6: break ;
7: **end if**
8: **end for**
9: **for** $l' := l \rightarrow 1$ **do**
10: **if** $F_{PC}(p, l'-1, r') \wedge valid(\vec{f}_{p+l'-1})$ 是可满足的 **then**
11: break;
12: **else if** $l' \equiv 1$ **then**
13: $l' := l' - 1$;
14: break;
15: **end if**
16: **end for**
17: **return** $< l', r' >$

3.5.1 压缩 l 和 r

由于我们在算法5.3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5.3，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5.4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1，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。

3.5.2 另一种可能的算法结构

在上述讨论中，我们在算法3.3中同步增加 p, l 和 r 以找到流控变量，然后在算法5.4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3.7.6 中指出，同步增加 p, l 和 r 然后使用算法5.4 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。

3.6 产生解码器函数

在小节3.3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。

3.6.1 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：

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & f_{p+l} \equiv 1 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \end{array} \right\} \tag{3.19}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & f'_{p+l} \equiv 0 \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.20}$$

显然 $\phi_A \wedge \phi_B$ 等价于等式 (5.1) 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[11] 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。

3.6.2 产生 $\vec{d}$ 的解码函数

假设 $valid(\vec{f})$ 是被算法5.3推导出来的谓词。为每个 $d \in \vec{d}$，我们定义以下两个公式：

$$\phi'_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & d_{p+l} \equiv 1 \\ \wedge & valid(\vec{f}_{p+l}) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i}_m) \end{array} \right\} \tag{3.21}$$

$$\phi'_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & d'_{p+l} \equiv 0 \\ \wedge & valid(\vec{f'}_{p+l}) \\ \wedge & \bigwedge_{m=0}^{p+l+r} assertion(\vec{i'}_m) \end{array} \right\} \tag{3.22}$$

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[11] 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3.6.1 中的增量求解机制加速该算法。

3.7 实验结果

我们使用 OCaml 语言实现了所有算法。并使用 MiniSat 1.14[17] 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。

3.7.1 测试集

表3.1 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[5].
2. Liu et al. [7].

表3.1 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[29]，脚本为”strash; dsd; strash; dc2; dc2; dch; map”。该脚本来自于[7]。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[7] 作比较。

表3.1 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[5] 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 3.7.2,3.7.3 和3.7.4中描述。

表 3.1 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[5]并有流控的测试集	PCIE2	10 / 11	22 / 149	326	PCIE 2.0 [28]	小节 3.7.2
	XGXS	10 / 10	17 / 249	572	10Gb 以太网 clause 48[27]	小节 3.7.3
	T2Eth	14 / 14	53 / 427	947	UltraSPARC T2 的以太网模块	小节 3.7.4
来自于[5]但没有流控的测试集	XFI	72 / 66	72 / 5086	12381	10Gb 以太网 clause 49[27]	在小节3.7.5 中比较我们的算法和 Liu[7]
	SCRAM-BLER	64/64	58/353	1034	增加数据中的 01 翻转	
来自于[7] 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	

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3.7.5中描述。

我们还进行了下列额外的实验：

在小节3.7.6中，我们将比较下列两种不同算法的时间开销：

1. 在算法5.3中同时增长 p, l 和 r，然后在算法5.4中压缩他们的值。

2. 在算法5.3中使用三个嵌套的循环分别增长 p, l 和 r。

在小节3.7.7中，我们将比较在算法5.4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 3.7.8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。

3.7.2 PCI Express 2.0 编码器

该编码器遵从 PCI Express 2.0 标准[28]。在删除了所有的注释和空行之后，其源代码包含 259 行 verilog。

表3.2给出了所有输入和输出的描述。根据 8b/10b 编码机制的描述[30], 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5.1 在 0.475 秒内识别出了流控向量 $\vec{f} := CNTL\_TXEnable\_P0$。而后算法5.3 在 1.22 秒内推导出了 $valid(\vec{f}) := CNTL\_TXEnable\_P0$。最后算法5.4 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

表 3.2 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 3.3　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

仅当 $CNTL\_TXEnable\_P0 \equiv 0$ 成立，也就是 $TXDATA$ 和 $TXDATAK$ 无效时，输出 $HSS\_TXELECIDLE$ 为 1. 因此，解码器将使用 $HSS\_TXELECIDLE$ 来唯一决定 $CNTL\_TXEnable\_P0$。

3.7.3　10G 以太网编码器 XGXS

该编码器 XGXS 遵从 IEEE 802.3 标准的[27] 的 clause 48 。在删除空行和注释后，其包含 214 行 verilog。

表3.3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[30] ，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[30]，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5.1 在 0.31 秒内识别了流控向量 $\vec{f} := bad\_code$。然后算法5.3 在 0.95 秒内推导了谓词 $valid(\vec{f}) := bad\_code$。再次算法5.4 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*。

3.7.4　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[27] 的 clause 36。在删除空行和注释之后，它包含 864 行 verilog 源代码。

表3.4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[30]，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3.5所示。很明显控制字符和流控机制被混合在 *tx_enc_ctrl_sel* 中。表3.5 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5.1 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5.3 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5.4 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3.5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_*10*bdata*。因此解码器总能从 *tx_*10*bdata* 恢复出 *tx_enc_ctrl_sel*。

3.7.5 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3.6 针对不具备流控机制的编码器比较了我们的算法和[7] 的算法。

对于从 XFI 到 HM(15,11) 的 6 个 benchmark，我们有他们的源代码，而[7] 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[7] 的结果。

通过比较第二和第三列之和和第六列，可见[7] 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[7] 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[7]，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 [29] 的相应代码得到改善。

表 3.4　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3.5
	tx_en	1	传输使能
	tx_er	1	传输一个错误字符
输出	*tx_*10*bdata*	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 3.5　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

3.7.6　比较两种可能性：同时增长 p, l 和 r 或者单独增长

在算法3.3中，我们同时增加 p, l 和 r ，并在算法5.4中压缩他们的冗余值。我们称其为 A1 方案。

小节3.5.2 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3.7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。

表 3.6　比较我们的算法和[7] 的算法

名字	我们的算法				[7]		
	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 3.7　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

从小节3.5.2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5.4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。

3.7.7　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5.4 被用于在产生解码器之前压缩 l 和 r。表3.8展示了其效果。

第一列是 benchmark 名字。当算法5.4 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5.4 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5.4显著的压缩了 l 和 r。

表 3.8　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5.4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

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。

3.7.8　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3.9 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。

表 3.9　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

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。

3.8 相关工作

3.8.1 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.[6] 独立的发现了如何通过检测环来得到停机的算法。而我们[5] 和 Liu et al.[6] 独立的发现可以通过 Craig 插值加速解码器的生成。

我们[5] 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5.3的特例。而本文算法5.3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.[8] 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[20, 21]，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。

3.8.2 程序求反

文献[31] 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[32]，只能处理非常小的程序和非常简单的语法。

Glück et al. [33] 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[34] 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。

3.8.3 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[35, 36] 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[37] 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。

3.8.4 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[38] 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[39] 和[40] 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[41] 和[42, 43] 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[44] 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 [18] 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[13] 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[45] 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。

3.8.5 基于 Craig 插值的逻辑综合算法

在文献[46, 47] 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[5] 以找到所有可能的解码器。

在文献[48] 中，Craig 插值被用于产生 ECO。

在文献[13] 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。

3.9 结论

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 [28] 和以太网[27]。

trying to cite some papers

[49] [50]

[51]

[52]

第四章　面向流水线的对偶综合

4.1　引言

在通讯和多媒体芯片设计项目中，最困难的一个工作是针对特定的协议，如以太网 [27] 和 PCI Express [28]，设计相应的编码器和解码器. 其中编码器负责将输入 $\vec{i}$ 映射至输出 $\vec{o}$，而解码器则负责从 $\vec{o}$ 中恢复 $\vec{i}$。对偶综合 [1, 3, 5–8] 假设 $\vec{i}$ 总能够被 $\vec{o}$ 的一个有限序列唯一决定，以自动产生相应的解码器。其中，解码器的布尔函数可以使用 Jiang et al. [13] 提出的算法，该算法基于 Craig 插值 [10]。

通过研究现有的工业界编码器，我们发现他们都含有流水线结构以提高运行频率。

一个带有流水线的简单编码器如图**??**a) 所示。他的第一级流水线 $s\vec{t}g^0$ 包含数个寄存器。其中输入变量 $\vec{i}$ 被用于计算该级流水线 $s\vec{t}g^0$，而流水线 $s\vec{t}g^0$ 则被用于计算 $\vec{o}$。根据该结构，$s\vec{t}g^0$ 能够被 $\vec{o}$ 唯一决定，而 $\vec{i}$ 能够被 $s\vec{t}g^0$ 唯一决定。

因此，一个由人类程序员设计的合理解码器, 应当如图**??**b) 所示，从 $\vec{o}$ 中使用组合逻辑 C^1 恢复 $s\vec{t}g^0$，并进一步使用组合逻辑 C^0 从 $s\vec{t}g^0$ 中恢复 $\vec{i}$。在此类解码器中，关键路径被流水线级 $s\vec{t}g^0$ 切断，以改善时序。

然而，目前所有的对偶综合算法 [3, 5–8] 均使用 Jiang 提出的基于 [10] 的算法 [13]。如图**??**c) 所示，这些算法从 $\vec{o}$ 中使用一个大型组合逻辑 $C^0 * C^1$ 直接恢复 $\vec{i}$, 这使得他们变得不必要的很慢，因为没有流水线寄存器切断这段复杂逻辑。

为了产生如图**??**b) 所示的解码器，本文提出了一个新颖的算法。首先找到编码器中每一个流水线级 $s\vec{t}g^j$ 中的寄存器，然后特征化每一个流水线级 $s\vec{t}g^j$ 的布尔函数，已从下一个流水线级 $s\vec{t}g^{j+1}$ 或输出 $\vec{o}$ 之中恢复 $s\vec{t}g^j$。最终特征化 $\vec{i}$ 的布尔函数以葱第一个流水线级 $s\vec{t}g^0$ 中恢复 $\vec{i}$。

在复杂的工业界实际编码器，如 PCI Express [28] 和以太网 [27]，上的实验表明，该算法总能够产生带有流水线级的快速解码器。.

本文剩余的内容安排如下. 第5.2 节介绍了背景知识; 第5.4 节推导流水线结构，而第5.5 节特征化每一个流水线级和输入的布尔函数; 第5.6 和5.7 节给出实验结果和相关工作; 最后，第5.8 节给出结论。

4.2　背景知识

4.2.1　命题逻辑可满足

布尔集合为 $\mathbb{B} = \{0, 1\}$。变量向量表示为 $\vec{v} = (v, \dots)$。向量 $\vec{v}$ 中的变量个数表示为 $|\vec{v}|$。若一个变量 v 是 $\vec{v}$ 的成员，则记为 $v \in \vec{v}$; 否则 $v \notin \vec{v}$. $v \cup \vec{v}$ 是包含 v 和 $\vec{v}$

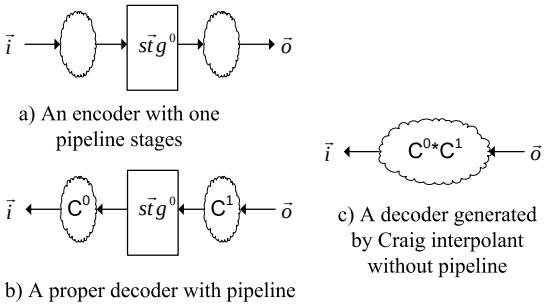


图 4.1　带有流水线的编码器和解码器

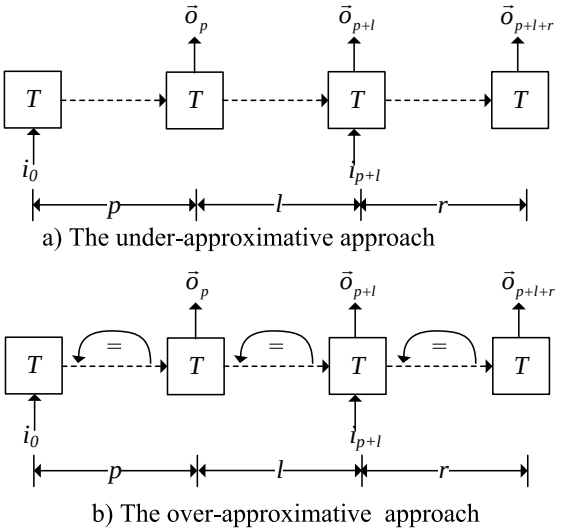


图 4.2　下估计和上估计算法

的所有成员的新向量。$\vec{v}/\vec{w}$ 是一个包含 $\vec{v}$ 的所有成员但是不包含 $\vec{w}$ 的任意成员的新向量。$\vec{a} \cup \vec{b}$ 是包含 $\vec{a}$ 和 $\vec{b}$ 的所有成员的新向量。$\vec{v}$ 的赋值集合记为 $[\![\vec{v}]\!]$，比如，$[\![(v_1, v_2)]\!] = \{(0,0),(0,1),(1,0),(1,1)\}$。

对于变量集合 V 上的公式 F，起命题逻辑可满足 (SAT) 问题是指如何找到 V 的一个赋值函数 $A : V \to \mathbb{B}$，使得 F 能够被赋值为 1。若 A 存在，则 F 是可满足的；否则，是不可满足的。

对于公式 ϕ_A 和 ϕ_B，若 $\phi_A \wedge \phi_B$ 不可满足，则存在公式 ϕ_I 仅包含 ϕ_A 和 ϕ_B 的共同变量，并使得 $\phi_A \Rightarrow \phi_I$ 且 $\phi_I \wedge \phi_B$ 不可满足。ϕ_I 称为 ϕ_A 相对于 ϕ_B 的 Craig 插值 [10]。ϕ_I 可以使用 McMillan 提出的算法 [11] 进行计算，并应用于从一个布尔关系特征化出一个布尔函数 [13]。

4.2.2 游有限状态机

编码器使用有限状态机模型 (FSM) $M = (\vec{s}, \vec{i}, \vec{o}, T)$，其中 $\vec{s}$ 为状态变量向量，$\vec{i}$ 为输入变量向量，$\vec{o}$ 为输出变量向量，$T : [\![\vec{s}]\!] \times [\![\vec{i}]\!] \to [\![\vec{s}]\!] \times [\![\vec{o}]\!]$ 是一个状态迁移函数，用于从当前状态和输入变量向量，计算出下一状态和输出变量向量。

M 的行为可以通过展开迁移关系进行推导。在第 n 步的 $s \in \vec{s}$，$i \in \vec{i}$ 和 $o \in \vec{o}$ 分别表示为 s_n, i_n 和 o_n。另外，在第 n 步的单个状态，输入和输出变量分别表示为 $\vec{s}_n$，$\vec{i}_n$ 和 $\vec{o}_n$。一个路径是指 $< \vec{s}_n, \ldots, \vec{s}_m >$，其中对于 $n \le j < m$，有 $\exists \vec{i}_j \vec{o}_j (\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$。一个环是一个路径 $< \vec{s}_n, \ldots, \vec{s}_m >$ 满足 $\vec{s}_n \equiv \vec{s}_m$。

4.2.3 用于确定一个输入变量能否被输出向量的有限序列唯一决定的停机算法

该算法 [3] 迭代的展开迁移函数。对于每次迭代，其分别调用一个下估计和一个上估计算法已给出确定的答案。这两个算法分别在5.2.3.1 和5.2.3.2中给出。我们酱紫啊4.2.3.3 中指出这两者将最终收敛。

4.2.3.1 下估计算法

如图5.3a) 所示，在展开的迁移函数序列上，输入变量 $i \in \vec{i}$ 能够被唯一决定的充分条件是，存在 p，l 和 r，使得对于输出向量序列 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 的任意值，i_{p+l} 不能同时为 0 和 1。折等价于公式 (5.1) 中定义的 $F_{PC}(p, l, r)$ 的不可满足性。

$$
F_{PC}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \end{array} \right\} \tag{4.1}
$$

其中，p 是前置状态序列的长度，l 和 r 是两个用于唯一决定输入 i_{p+l} 的输出变量序列 $< \vec{o}_{p+1}, \ldots, \vec{o}_{p+l} >$ 和 $< \vec{o}_{p+l+1}, \ldots, \vec{o}_{p+l+r} >$ 的长度。行 2 对应于图5.3a) 中的路径，行 3 是他的一个拷贝。他们的长度是一样的。行 4 强制两者的输出序列具有相同的赋值，而行 5 强制他们的输入 i_{p+l} 不同。

根据公式 (5.1)，对于 $p' \ge p$，$l' \ge l$ 和 $r' \ge r$，$F_{PC}(p', l', r')$ 的短句集合是 $F_{PC}(p, l, r)$ 的超集。因此, $F_{PC}(p, l, r)$ 的不可满足性可以扩展到任意更大的 $p' \ge p$，$l' \ge l$ 和 $r' \ge r$。

命题 4.1: *若 $F_{PC}(p, l, r)$ 不可满足，则对任意更大的 p, l 和 r，i_{p+l} 可以被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。*

算法 4.1 *CheckUniqueness*(i): 检查 $i \in \vec{i}$ 能否被 $\vec{o}$ 唯一决定的停机算法

```
1: 输入: 输入变量 i ∈ i⃗.
2: 输出: i ∈ i⃗ 是否能够被 o⃗ 唯一决定,p, l 和 r 的取值
3: p:= 0; l:= 0; r:= 0
4: while 1 do
5:     p++; l++; r++
6:     if F_PC(p, l, r) 不可满足 then
7:         return (yes, p, l, r);
8:     else if F_LN(p, l, r) 可满足 then
9:         return (no, p, l, r);
10:    end if
11: end while
```

4.2.3.2 上估计算法

若上一节定义的 $F_{PC}(p,l,r)$ 可满足，则存在两种可能性:

1. i_{p+l} 可以在更大的 p, l 和 r 时，被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定;
2. i_{p+l} 不能被任何 p, l 和 r 情况下的 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。

如果是第一种情况，则通过迭代的增大 p, l 和 r，$F_{PC}(p,l,r)$ 总能够变成不可满足。而如果是第二种情况，则该算法不停机。

因此, 为了得到停机算法我们需要区分上述两种情形。该方法如图5.3b) 所示，该图类似于图5.3a)，不过增加了三个新的约束用于检测在 $<\vec{s}_0,\ldots,\vec{s}_p>$,$<\vec{s}_{p+1},\ldots,\vec{s}_{p+l}>$ 和 $<\vec{s}_{p+l+1},\ldots,\vec{s}_{p+l+r}>$ 上的环。其形式化的定义如公式 (5.2) 其中最后三行对应于新增加的三个环检测约束。

$$F_{LN}(p,l,r) := \left\{\begin{array}{ll} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1}\bigvee_{y=x+1}^{p}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1}\bigvee_{y=x+1}^{p+l}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1}\bigvee_{y=x+1}^{p+l+r}\{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array}\right\} \quad (4.2)$$

当 $F_{LN}(p,l,r)$ 可满足时，则 i_{p+l} 不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。更重要的是，通过展开这三个环我们可以把 $F_{LN}(p,l,r)$ 的可满足性推广到更大的 p, l 和 r 上。这意味着:

命题 4.2: 如果 $F_{LN}(p,l,r)$ 可满足，则 i_{p+l} 对于任意更大的 p, l 和 r，不能被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。

其正确性证明的细节见 [3]。

4.2.3.3 完全算法

基于 Propositions 4.1 和4.2，我们有如下算法4.1。该算法搜索 p, l 和 r 使得输入变量 i_{p+l} 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

1. 一方面，如果存在这样的 p, l 和 r，则令 $p' := max(p,l,r), l' := max(p,l,r)$ 和 $r' := max(p,l,r)$。从 Propositions 4.1，我们知道 $F_{PC}(p', l', r')$ 不可满足。因此 $F_{PC}(p,l,r)$ 总能在行5称为不可满足；

2. 另一方面，如果不存在 p, l 和 r，则 p, l 和 r 总会变得比最长的无环路径更长，这意味着在 $< \vec{s}_0, \ldots, \vec{s}_p >, < \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ 和 $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$ 上存在环。这将使得 $F_{LN}(p,l,r)$ 在行7变得可满足。

两种情况都将导致上述算法停机。其正确性和停机性证明的细节见 [3] 。

4.3 推导编码器的流水线结构

4.3.1 流水线的一般性模型

如图5.5所示，我们假设编码器有 n 几流水线。如果我们把组合逻辑 C^j 视为一个函数，则编码器可以用下列的公式表示：

$$\begin{array}{rcll} \vec{stg}^{\,0} & := & C^0(\vec{i}) & \\ \vec{stg}^{\,j} & := & C^j(\vec{stg}^{\,j-1}) & 1 \le j \le n-1 \\ \vec{o} & := & C^n(\vec{stg}^{\,n-1}) & \end{array} \tag{4.3}$$

因此, 每个 C^j 可以视为一个小型的编码器用于从 $\vec{stg}^{\,j-1}$ 或 $\vec{i}$ 中计算 $\vec{stg}^{\,j}$ 或 $\vec{o}$。

在本文的剩余部分，上标始终表示特定的流水线级，, 而下标则如小节5.2.2所述，表示在展开的迁移关系序列中的步数。例如, $\vec{stg}^{\,j}$ 是第 j 个流水线级。而 $\vec{stg}^{\,j}_i$ 则是该第 j 流水线级在展开的迁移函数序列中的第 i 步的值。

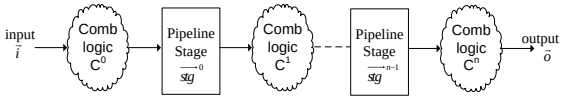


图 4.3 编码器的一般性结构

算法 4.2 *RemoveRedundancy*(p, l, r)

1: **for** $r' := r \to 0$ **do**
2: **if** $r' \equiv 0$ or $F_{PC}(p, l, r'-1)$ 对于某些 $i \in \vec{i}$ 是可满足的 **then**
3: break;
4: **end if**
5: **end for**
6: **return** r';

4.3.2 推导 p, l 和 r

在推导之前，我们首先使用算法4.1 以得到 p, l 和 r 。

因为存在多于一个 $i \in \vec{i}$，我们需要为每一个 $i \in \vec{i}$ 使用算法4.1 以得到他们对用的 p, l 和 r 。

然后我们将最终的 p,l 和 r 设置为所有 $i \in \vec{i}$ 中最大的 p, l 和 r 。根据公式5.1，这些 p, l 和 r 能够使得 $< o_p, \ldots, o_{p+l+r} >$ 唯一决定所有 $i_{p+l} \in \vec{i}_{p+l}$。

4.3.3 压缩 r 和 l

算法4.1 同步增长 p, l 和 r ，因此在 l 和 r 中可能存在冗余。因此我们需要首先在算法5.4中压缩 r。

在行1，当 $F_{PC}(p, l, r'-1)$ 可满足时，则 r' 是最有一个使得 $F_{PC}(p, l, r')$ 不可满足的，我们将其直接返回。另一方面，当 $r' \equiv 0$，$F_{PC}(p, l, 0)$ 已经在上一次迭代中被测试过, 且结果必然是不可满足。在这种情况下我们返回 0。

如此, 我们从算法5.4获得了一个压缩后的 r ，它能使 $\vec{i}_{p+l}$ 被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。

我们进一步要求：

1. 如图5.6所示，l 可以被压缩为 0, 这意味着 $\vec{i}_p$ 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+r} >$ 唯一决定，也就是说，所有的将来输出。

2. 上述的序列 $< \vec{o}_p, \ldots, \vec{o}_{p+r} >$ 可以被进一步压缩为 $\vec{o}_{p+r}$. 这意味着 $\vec{o}_{p+r}$ 是在恢复 $\vec{i}_p$ 时唯一被需要的输出。

检查上述两个要求等价于下式的不可满足：

图 4.4　从压缩的输出序列中恢复输入

$$F'_{PC}(p,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & i_p \equiv 1 \wedge i'_p \equiv 0 \end{array} \right\} \tag{4.4}$$

该等式看起来似乎远强于等式 (5.1)。我们将在实验结果中指出，该等式总能称为不满足。

4.3.4　推导流水线

基于上述的 p 和 r, 我们将上述公式 (5.13) 中的 F'_{PC} 推广到下述公式以便能够检查 v 在第 j 步是否能够被 $\vec{w}$ 在第 k 步唯一决定。现在 v 和 $\vec{w}$ 可以使输入，输出和状态变量向量。

$$F''_{PC}(p,r,v,j,\vec{w},k) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{w}_k \equiv \vec{w'}_k \\ \wedge & v_j \equiv 1 \wedge v'_j \equiv 0 \end{array} \right\} \tag{4.5}$$

很显然, 当 $F''_{PC}(p,r,v,j,\vec{w},k)$ 不可满足时，$\vec{w}_k$ 能够唯一决定 v_j.

最后一集流水线 $\vec{stg}^{n-1}$ 就是所有能够被在第 $p+r$ 步的 $\vec{o}$ 唯一决定的 $s \in \vec{s}$。其定义为:

$$\vec{stg}^{n-1} := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,p+r,\vec{o},p+r) \\ is\ unsatisfiable \end{array} \right\} \tag{4.6}$$

类似的, 对于 $0 \le j \le n-2$, $\vec{stg}^{j}$ 在第 $j-((n-2)-(p+r-1))$ 步可以被 $\vec{stg}^{j+1}$ 在第 $j-((n-2)-(p+r-1))+1$ 步唯一决定。我们可以定义 $\vec{stg}^{j}$ 为:

$$\begin{array}{lll} S & := & \vec{s} / \bigcup_{j<k\leq n-2} \vec{stg}^{k} \\ D & := & (n-2)-(p+r-1) \end{array} \tag{4.7}$$

$$\vec{stg}^{j} := \left\{ s \in S \mid F''_{PC}(p,r,s,j-D,\vec{stg}^{j+1},j-D+1) \right\} \tag{4.8}$$

基于等式 (4.6) 和 (4.8), 所有的流水线几都能够被推导出来。

4.3.5 推导唯一决定输入的流水线级

根据图5.5, 定义于 (4.8) 的 $\vec{stg}^{0}$ 是唯一决定 $\vec{i}$ 的流水线级。

然而在实际的解码器中，并不一定是这种情形。因此我们需要从 0 到 $n-1$，搜索最小的 j 使得 $\vec{i}$ 能够被 $\vec{stg}^{j}$ 唯一决定。即, 最小的能够使得 $F''_{PC}(p,r,i,p,\vec{stg}^{j},j-D)$ 对所有 $i \in \vec{i}$ 不可满足的 j。其中 D 定义于等式 (5.16)。

4.4 特征化输入向量和流水线级的布尔函数

4.4.1 特征化最后一个流水线级的布尔函数

根据等式 (4.6), 每个寄存器 $s \in \vec{stg}^{n-1}$ 都能够被 $\vec{o}$ 在第 $p+r$ 步唯一决定。即, $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ 不可满足，且可以被划分为:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+r} \equiv 1 \end{array} \right\} \tag{4.9}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & s'_{p+r} \equiv 0 \end{array} \right\} \tag{4.10}$$

由于 $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ 等价于 $\phi_A \wedge \phi_B$, 因此 $\phi_A \wedge \phi_B$ 也是不可满足的。而 ϕ_A 和 ϕ_B 的共同变量集合是 $\vec{o}_{p+r}$。

根据 [13], ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以构造出来，其中仅包含 $\vec{o}_{p+r}$，并覆盖所有使 $s_{p+r} \equiv 1$ 得 $\vec{o}_{p+r}$ 的赋值。同时, $\phi_I \wedge \phi_B$ 不可满足，这意味着 ϕ_I 不能使 $s_{p+r} \equiv 0$。

因此, ϕ_I 能作为解码器的布尔函数以从 $\vec{o}$ 中恢复 $s \in \vec{stg}^{n-1}$ 。

4.4.2 特征化其他流水线级的布尔函数

类似于上一小节, 我们可以将公式 (4.8) 中的不可满足公式 $F''_{PC}(p, r, s, j - D, \vec{stg}^{j+1}, j - D + 1)$ 划分如下:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & s_{j-D} \equiv 1 \end{array} \right\} \tag{4.11}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{j+1}_{j-D+1} \equiv \vec{stg'}^{j+1}_{j-D+1} \\ \wedge & s'_{j-D} \equiv 0 \end{array} \right\} \tag{4.12}$$

ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 能够被构造出来，并作为从 $\vec{stg}^{j+1}$ 恢复 $s \in \vec{stg}^{j}$ 的布尔函数。

4.4.3 特征化输入变量的布尔函数

根据小节5.4，我们找到了使得 $F''_{PC}(p, r, i, p, \vec{stg}^{j}, j - D)$ 对所有 $i \in \vec{i}$ 都不满足的 j，其中 D 在的公式 (5.16) 中定义。$F''_{PC}(p, r, i, p, \vec{stg}^{j}, j - D)$ 是不可满足的并可划分为:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & i_p \equiv 1 \end{array} \right\} \tag{4.13}$$

表 4.1　Benchmarks and experimental results

Names	编码器				由 [3] 产生的解码器			本文产生的解码器			
	# in/out	# reg	面积	编码器描述	运行时间	延迟 (ns)	面积	运行时间	延迟 (ns)	面积	寄存器个数
pcie	10/11	23	326	PCIE 2.0 [28]	0.37	7.20	624	3.57	5.89	652	9/12
xgxs	10/10	16	453	以太网 clause 48 [27]	0.21	7.02	540	1.57	5.93	829	13
t2eth	14/14	49	2252	以太网 clause 36 [27]	12.7	6.54	434	47.2	6.12	877	8/8/10/20
scrambler	64/64	58	1034	inserting 01 flipping	no pipeline stages found						
xfi	72/66	72	7772	以太网 clause 49 [27]							

$\phi_B :=$

$$\left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{\,j}_{j-D} \equiv \vec{stg'}^{\,j}_{j-D} \\ \wedge & i'_p \equiv 0 \end{array} \right\} \quad (4.14)$$

和上一小节类似, ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以作为从 $\vec{stg}^{\,j}$ 中恢复 $i \in \vec{i}$ 的布尔函数。

4.5　实验结果

我们在 OCaml 语言中实现了上述算法，并使用 MiniSat 1.14 [17] 求解相应的公式。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。

表5.1 给出了本文使用的 benchmarks 。第 2 和 3 列分别给出了每个 benchmark 的输入, 输出和寄存器数量。第 4 列将这些编码器映射至 LSI10K 库的面积。本文中，所有的面积和延时都是在相同的设定下得到的。

第 6 到 8 列分别给出了 [3] 算法在产生误流水线的解码器时的运行时间，延迟和面积。而第 9 到 11 列分别给出了本文算法的运行时间，延迟和面积。而最后一列给出了每一级流水线包含的寄存器个数。

比较第 7 和 10 列可以看到解码器的延迟得到了较大的改善，而最后一列指出确实存在很深的流水线，其中 t2ether 包含 4 级流水线。

有一点非常有意思的是, 两个最大的 benchmarks scrambler 和 xfi 没有检测到流水线。我们研究了其代码，并确认了这一点。他们的面积如此之大是因为使用了 64 到 72 位宽的数据路径。

4.6 相关工作

沈等 [1] 提出了第一个对偶综合算法。他通过迭代的展开迁移函数来检测解码器的存在性。并通过遍历可满足赋值了产生解码器函数。但是该算法是不停机的，并且在产生解码器时非常慢。

沈 et al.[3] 和刘 at al.[6] 分别独立处理了停机问题，方法是在状态序列中检测环形路径。而解码器构造中运行时间太长的问题则在 [5, 6] 中通过 Craig 插值 [13] 解决。

沈 et al. [5] 推导了能够使得解码器存在的配置断言。

屠 et al.[8] 提出了一个突破性的算法能够在构造解码器时考虑整个无限的输入历史。

4.7 结论

本文提出了第一个能够处理流水线的对偶综合算法。实验结果表明，本算法能够针对多个复杂的实际工业界编码器，正确的推导流水线结构并生成对应的流水线解码器。.

第五章　面向流控机制和流水线的对偶综合

5.1　引言

在通讯和多媒体芯片设计项目中，一个最困难的工作之一是为不同的协议设计编码器和解码器。其中编码器负责将输入向量 $\vec{i}$ 映射到输出向量 $\vec{o}$，而解码器负责从 $\vec{o}$ 中恢复 $\vec{i}$。对偶综合 [1, 3, 5–8] 假设 $\vec{i}$ 总能够被 $\vec{o}$ 唯一决定，并自动产生相应的解码器。

然而, 许多编码器中采用的留空机制 [9] 不能满足该要求。如图5.1a) 所示，当接收器无法跟上发送器时，该机制通过发送空闲字符 I 以防止快速发送器充爆慢速接收器。如图5.1b) 所示，空闲字符 I 只能唯一决定 $\vec{i}$ 的一部分而非全部，我们称之为流控向量 $\vec{f}$。而正常的编码结果 D_i 能唯一决定所有输入包括流控向量 $\vec{f}$ 和数据向量 $\vec{d}$。

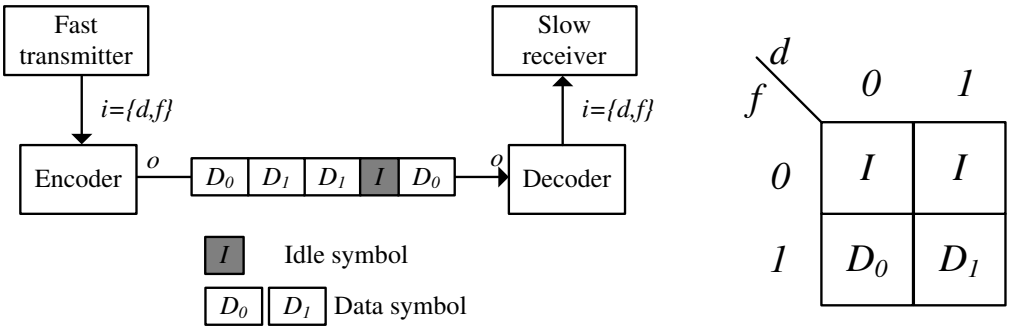
秦 et al. [12] 首次提出了能够处理流控机制的对偶综合算法。该算法首先找到所有的能够被 $\vec{o}$ 唯一决定的 $i \in \vec{f}$。然后推导一个能使得 $\vec{d}$ 被 $\vec{o}$ 唯一决定的谓词 $valid(\vec{f})$。

同时，如图5.2所示，许多编码器包含流水线级 $s\vec{t}g^j$ 已将关键的数据路径划分为多个子段 C^j, 这样有助于提高性能。类似于 $\vec{i}$, 每个流水线级 $s\vec{t}g^j$ 也可以划分为留空向量 $\vec{f}^j$ 和数据向量 $\vec{d}^j$。

然而由秦 et al. 的算法 [12] 产生的解码器并不包含流水线。这使得其运行速度远低于相应的编码器。为了解决该问题，本文提出了一个全新的算法以为此类编码器产生带有流控机制和流水线的解码器。该算法首先使用秦 et al. [12] 的算法来寻找 $\vec{f}$ 并推导 $valid(\vec{f})$。然后分别通过强制和不强制 $valid(\vec{f})$，已从所有寄存器集合中找到每一个寄存器级 $s\vec{t}g^j$ 的 $\vec{d}^j$ 和 $\vec{f}^j$。最后通过 Jiang et al. [13] 的算法特征化 $s\vec{t}g^j$ 和 $\vec{i}$ 的布尔函数。

实验结果表明，该算法能够为多个工业界的真实编码器正确的产生带有流控和流水线的解码器。

本文剩余部分如下组织。小节5.2 介绍相关的背景知识；小节5.3 介绍本文算法的整体结构；小节5.4 找到每一个流水线级 $s\vec{t}g^j$ 中的 $\vec{f}^j$ 和 $\vec{d}^j$ 。小节5.5 为 $s\vec{t}g^j$ 和 $\vec{i}$ 特征化布尔函数。小节5.6 和5.7 分别给出实验结果和相关工作。最后小节5.8 给出结论。



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 5.1　带有流控机制的编码器

5.2　背景知识

5.2.1　命题逻辑可满足

布尔集合记为 $\mathbb{B} = \{0, 1\}$。多个变量组成的向量记为 $\vec{v} = (v, \dots)$。$\vec{v}$ 中的变量个数记为 $|\vec{v}|$。若一个变量 v 是 $\vec{v}$ 的成员，则记为 $v \in \vec{v}$；否则记为 $v \notin \vec{v}$。对于一个变量 v 和一个向量 $\vec{v}$，若 $v \notin \vec{v}$，则一个同时包含 v 和所有 $\vec{v}$ 的成员的新变量记为 $v \cup \vec{v}$。若 $v \in \vec{v}$，则一个包含 $\vec{v}$ 的所有成员，但是不包含 v 的新变量记为 $\vec{v} - v$。对于两个向量 $\vec{a}$ 和 $\vec{b}$，则同时包含 $\vec{a}$ 和 $\vec{b}$ 的所有成员的新向量记为 $\vec{a} \cup \vec{b}$。

对于变量集合 V 上的公式 F，其命题逻辑可满足问题 (SAT) 的目的在于巡展赋值函数 $A : V \to \mathbb{B}$，使得 F 能够取值为 1。若 A 存在则 F 是可满足的；否则是不可满足的。

对于两个布尔命题逻辑公式 ϕ_A 和 ϕ_B，若 $\phi_A \wedge \phi_B$ 不可满足，则存在仅引用 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 称为 ϕ_A 相对于 ϕ_B 的 Craig 插值 [10]。可以使用 McMillan 算法 [11] 产生该插值。

5.2.2　有限状态机

编码器使用有限状态机 $M = (\vec{s}, \vec{i}, \vec{o}, T)$ 作为模型，其中包含状态向量 $\vec{s}$。输入向量 $\vec{i}$，输出向量 $\vec{o}$，状态迁移函数 $T : \vec{s} \times \vec{i} \to \vec{s} \times \vec{o}$ 从当前状态向量和输入向量计算出下一状态向量和输出向量。

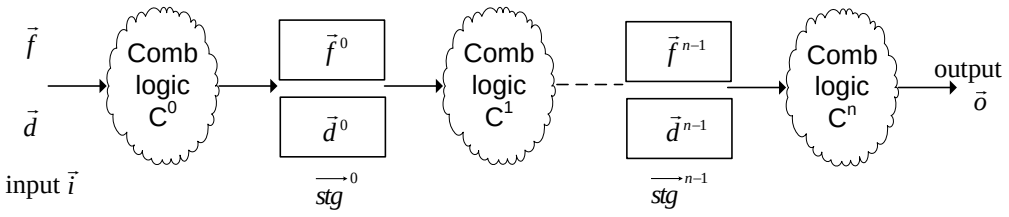


图 5.2　带有流水线和流控机制的编码器

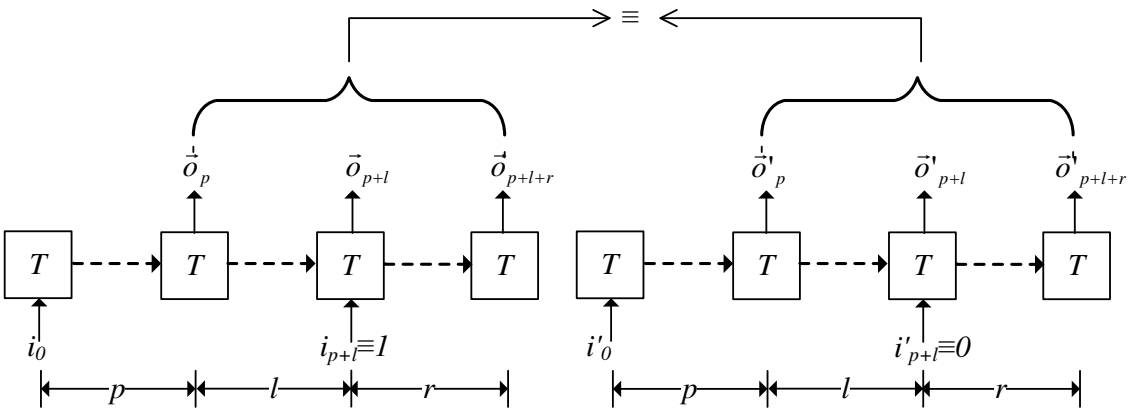


图 5.3　sound 和 complete 方法

M 的行为可以通过展开状态迁移函数进行推导。状态变量 $s \in \vec{s}$, 输入变量 $i \in \vec{i}$ 和输出变量 $o \in \vec{o}$ 在上述展开序列的第 n 步中分别记为 s_n, i_n 和 o_n。更进一步的，在第 n 步中的状态向量，输入向量和输出向量分别记为 $\vec{s}_n, \vec{i}_n$ 和 $\vec{o}_n$。一个路径是一个序列 $< \vec{s}_n, \ldots, \vec{s}_m >$ 使得对于所有 $n \leq j < m$，有 $\exists \vec{i}_j \vec{o}_j (\vec{s}_{j+1}, \vec{o}_j) \equiv T(\vec{s}_j, \vec{i}_j)$。而一个环是一个路径 $< \vec{s}_n, \ldots, \vec{s}_m >$ 使得 $\vec{s}_n \equiv \vec{s}_m$。

5.2.3　寻找 $\vec{f}$ 的停机算法

秦 et al. [12] 提出了一个寻找 $\vec{f}$ 的停机算法。该算法通过迭代的调用一个 sound 和一个 complete 的算法以最终得到收敛的答案。.

5.2.3.1　sound 算法

如图5.3a) 所示，在展开的迁移关系上，一个输入变量 $i \in \vec{i}$ 能够被唯一决定，是指存在 p, l 和 r，使得对于输出序列 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 的每一个取值，i_{p+l} 不能同时为 0 和 1。折等价于公式 (5.1) 中的 $F_{PC}(p, l, r)$ 的不可满足性。行 1 对应于图5.3a) 中的路径，而行 2 是其拷贝行 3 强制这两个路径的输出相等。而行 4 强制 i_{p+l} 不等。该算法是 sound 的因为当 (5.1) 不可满足 i 肯定是 $\vec{f}$ 的成员。

$$F_{PC}(p,l,r) := \left\{ \begin{array}{ll} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & i_{p+l} \equiv 1 \wedge i'_{p+l} \equiv 0 \end{array} \right\} \tag{5.1}$$

5.2.3.2　complete 算法

对于上述的 $F_{PC}(p, l, r)$ ，有两种可能性: **(1)**. 存在 p, l 和 r，使得 i_{p+l} 能够被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定; 或者 **(2)**. i_{p+l} 对于任意 p, l 和 r 都不能被唯一决定。

算法 5.1 Identifying the flow control vector $\vec{f}$

1: 输入：输入向量 $\vec{i}$.
2: 输出：$\vec{f} \subset \vec{i}$, 在此次搜索中找到的最大 p, l 和 r
3: $\vec{f} := \{\}; \vec{d} := \{\}; p := 0 ; l := 0 ; r := 0$
4: **while** $\vec{i} \neq \{\}$ **do**
5: 假设 $i \in \vec{i}$
6: $p++;\ \ l++;\ \ r++$
7: **if** $F_{PC}(p,l,r)$ 对于 i 不可满足 **then**
8: $\vec{f} := i \cup \vec{f}$;
9: $\vec{i} := \vec{i} - i$
10: **else if** $F_{LN}(p,l,r)$ 对于 i 可满足 **then**
11: $\vec{d} := i \cup \vec{d}$;
12: $\vec{i} := \vec{i} - i$
13: **end if**
14: **end while**
15: **return** $(\vec{f}, p, l, r)$

对于第一种情形，通过迭代的增加 p, l 和 r，$F_{PC}(p,l,r)$ 总能够变成不可满足。而对于第二种情形，该算法将永不停机。因此, 为了得到一个停机算法，我们需要如图5.3b) 所示的方法去检查第二种情形。该方法类似于5.3a)，但是在三个状态序列 $< \vec{s}_0, \ldots, \vec{s}_p >, < \vec{s}_{p+1}, \ldots, \vec{s}_{p+l} >$ 和 $< \vec{s}_{p+l+1}, \ldots, \vec{s}_{p+l+r} >$ 上增加了三个约束用于检测环。该方法形式化的定义于公式 (5.2) 中。其中最后三行即为我们新加的三个约束。该方法是 complete，因为当其是可满足的时候，我们可以通过展开这三个环来证明第二种情形并断定 $i \notin \vec{f}$。

$$F_{LN}(p,l,r) := \left\{ \begin{array}{ll} & F_{PC}(p,l,r) \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{5.2}$$

5.2.3.3 通过算法5.1找到 $\vec{f}$

在行7, 所有能够被唯一决定的输入 i 将被移到 $\vec{f}$。若 $F_{LN}(p,l,r)$ 在行9是可满足的，所有可满足的输入 i 将被移到 $\vec{d}$. 该算法的正确性和停机性证明请见 [12] 。

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

该算法也是由秦 et al. [12] 提出的。它首先给出算法5.2，该算法用于特征化一个函数，覆盖所有能够使得一个布尔关系满足的赋值集合。然后如图5.4所

算法 5.2 $CharacterizingFormulaSAT(R,\vec{a},\vec{b},t)$

1: 输入：布尔关系 $R(\vec{a},\vec{b},t)$
2: 输出：能够使得 $R(\vec{a},\vec{b},1)$ 可满足的 $FSAT_R(\vec{a})$
3: $FSAT_R(\vec{a}) := 0$;
4: **while** $R(\vec{a},\vec{b},1) \wedge \neg FSAT_R(\vec{a})$ 可满足 **do**
5: 假设 $A : \vec{a} \cup \vec{b} \cup \{t\} \rightarrow \{0,1\}$ 是一个可满足赋值;
6: $\phi_A(\vec{a}) := R(\vec{a}, A(\vec{b}), 1)$;
7: $\phi_B(\vec{a}) := R(\vec{a}, A(\vec{b}), 0)$;
8: 假设 $ITP(\vec{a})$ 是 ϕ_A 相对于 ϕ_B 的 Craig 插值;
9: $FSAT_R(\vec{a}) := ITP(\vec{a}) \vee FSAT_R(\vec{a})$;
10: **end while**
11: **return** $FSAT_R(\vec{a})$

示，算法5.2 被用于特征化函数 $\neg FSAT_{PC}(p,l,r)$，$valid(\vec{f})$ 的单调递增下估计，和 $\neg FSAT_{LN}(p,l,r)$, $valid(\vec{f})$ 的单调递减上估计。最终我们指出这两者将收敛到 $valid(\vec{f})$。

5.2.4.1 特征化使得一个布尔关系可满足的布尔函数

对于一个布尔关系 $R(\vec{a},\vec{b},t)$，有 $R(\vec{a},\vec{b},0) \wedge R(\vec{a},\vec{b},1)$ 不可满足。算法5.2 特征化一个布尔寒素 $FSAT_R(\vec{a})$，该函数覆盖且仅覆盖了能使得 $R(\vec{a},\vec{b},1)$ 可满足的所有 $\vec{a}$。行3 找到 $\vec{a}$ 的一个赋值，尚未被 $FSAT_R(\vec{a})$ 覆盖而且能够使得 $R(\vec{a},\vec{b},1)$ 可满足。行5, 6 和7 使用 McMillan 算法 [11] 将该赋值放大为 $ITP(\vec{a})$。行8 将 $ITP(\vec{a})$ 加入 $FSAT_R(\vec{a})$。

5.2.4.2 计算 $valid(\vec{f})$ 的单调递增下估计

通过将公式 (5.1) 中的 i 替换为算法5.1中推导的 $\vec{d}$，我们有:

$$F^d_{PC}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \bigwedge_{m=p}^{p+l+r}\vec{o}_m \equiv \vec{o'}_m \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \end{array} \right\} \tag{5.3}$$

若 $F^d_{PC}(p,l,r)$ 可满足，则 $\vec{d}_{p+l}$ 不能被 $< \vec{o}_p, \ldots, \vec{o}_{p+l+r} >$ 唯一决定。通过收集公式 (5.3) 的第三行，我们定义 $T_{PC}(p,l,r)$:

$$T_{PC}(p,l,r) := \left\{ \bigwedge_{m=p}^{p+l+r}\vec{o}_m \equiv \vec{o'}_m \right\} \tag{5.4}$$

通过将 $T_{PC}(p,l,r)$ 替换回 $F^{d}_{PC}(p,l,r)$，我们有：

$$F'^{d}_{PC}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{PC}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \end{array} \right\} \tag{5.5}$$

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：

$$\vec{a} := \vec{f}_{p+l} \tag{5.6}$$

$$\vec{b} := \vec{d}_{p+l} \cup \vec{d'}_{p+l} \cup \vec{s}_0 \cup \vec{s'}_0 \cup \bigcup_{0\leq x\leq p+l+r, x\neq(p+l)} (\vec{i}_x \cup \vec{i'}_x) \tag{5.7}$$

因此，向量 $\vec{a} \cup \vec{b}$ 包含所有步的输入向量 $<\vec{i}_0,\ldots,\vec{i}_{p+l+r}>$ 和 $<\vec{i'}_0,\ldots,\vec{i'}_{p+l+r}>$。它同时也包含两个初始状态 $\vec{s}_0$ 和 $\vec{s'}_0$。因此 $\vec{a}$ 和 $\vec{b}$ 能唯一决定 $F'^{d}_{PC}(p,l,r,t)$ 中 t 的取值。这意味着 $R(\vec{a},\vec{b},1) \wedge R(\vec{a},\vec{b},0)$ 是不可满足的。因此，对于 p, l 和 r 的特定组合，在 $\vec{f}_{p+l}$ 上定义且能够使 $F'^{d}_{PC}(p,l,r,1)$ 满足的函数可以通过使用 $F'^{d}_{PC}(p,l,r,t)$, $\vec{a}$ 和 $\vec{b}$ 调用算法定义如下：

$$FSAT_{PC}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{PC}(p,l,r,t),\vec{a},\vec{b},t) \tag{5.8}$$

如图5.4所示，$\neg FSAT_{PC}(p,l,r)$ 是 $valid(\vec{f})$ 的针对 p, l 和 r 单调递增的下估计。

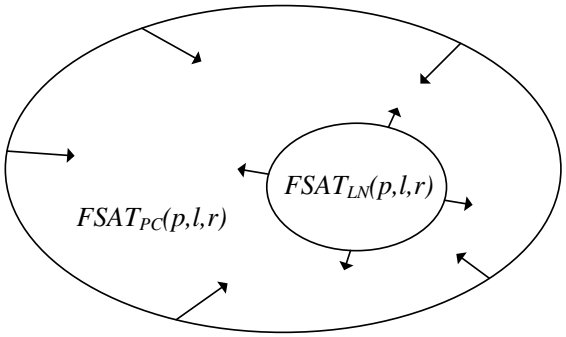


图 5.4　The monotonicity of $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$

算法 5.3 推导 $valid(\vec{f}_{p+l})$

1: p:= 0; l:= 0; r:= 0
2: **while** $\neg FSAT_{LN}(p,l,r) \wedge FSAT_{PC}(p,l,r)$ 可满足 **do**
3: p ++ ; l ++ ; r ++ ;
4: **end while**
5: **return** $\neg FSAT_{LN}(p,l,r)$

5.2.4.3 计算 $valid(\vec{f})$ 的单调递减上估计

类似的, 我们定义:

$$T_{LN}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{5.9}$$

$$F'^{d}_{LN}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & t \equiv T_{LN}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \end{array} \right\} \tag{5.10}$$

$$FSAT_{LN}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{LN}(p,l,r,t), \vec{a}, \vec{b}, t) \tag{5.11}$$

如图5.4所示，$\neg FSAT_{LN}(p,l,r)$ 是 $valid(\vec{f})$ 相对于 p, l 和 r. 的一个单调递减的上估计。

5.2.4.4 使用算法5.3推导 $valid(\vec{f})$

该算法迭代的增加 p, l 和 r，直到 $FSAT_{PC}(p,l,r)$ 和 $FSAT_{LN}(p,l,r)$ 收敛。其正确性和停机性见 [12] 。

5.3 算法框架

5.3.1 编码器的一般性模型

如图5.5所示，我们假设编码器包含 n 流水线级 $\vec{stg}^j$, 其中 $0 \leq j \leq n-1$。每一个流水线级 $\vec{stg}^j$ 能够被进一步划分为流控向量 $\vec{f}^j$ 和数据向量 $\vec{d}^j$。而输入向量 $\vec{i}$,

算法 5.4 压缩 r

```
1: for r' := r → 0 do
2:     if r' ≡ 0 or F_PC(p, l, r' − 1) ∧ valid(f⃗_{p+l}) 对于某些 i ∈ i⃗ 可满足 then
3:         break
4:     end if
5: end for
6: return r'
```

和 [12] 一样，也能被划分为留空向量 $\vec{f}$ 和数据向量 $\vec{d}$。如果将组合逻辑块 C^j 视为一个函数，则该编码器可以使用下列等式定义：

$$\begin{aligned} \vec{stg}^{\,0} &:= C^0(\vec{i}) \\ \vec{stg}^{\,j} &:= C^j(\vec{stg}^{\,j-1}) \quad 1 \le j \le n-1 \\ \vec{o} &:= C^n(\vec{stg}^{\,n-1}) \end{aligned} \tag{5.12}$$

在本文中，上标始终意味着流水线级, 而下标，, 如小节5.2.2指出, 始终意味着在展开的迁移关系序列中的步。例如, $\vec{stg}^{\,j}$ 是第 j 个流水线级。而 $\vec{stg}_i^{\,j}$ 该第 j 流水线级在第 i 步的取值。

5.3.2 算法框架

基于图5.5所示的编码器结构，我们算法的框架为：

1. 调用算法5.1 已将 $\vec{i}$ 划分为 $\vec{f}$ 和 $\vec{d}$。
2. 调用算法5.3 以推导能够使得 $\vec{d}$ 被唯一决定的 $valid(\vec{f})$ 和其对应的 p, l 和 r。
3. 在小节5.4, 找到每一个流水线级 $\vec{stg}^{\,j}$ 中的 $\vec{f}^j$ 和 $\vec{d}^j$ 。
4. 在小节5.5, 为每一个流水线级 $\vec{stg}^{\,j}$ 和输入向量 $\vec{i}$ 特征化布尔函数。

5.4 推导流水线结构

5.4.1 压缩 r 和 l

由于算法5.3 同时增加 p, l 和 r ，因此 l 和 r 存在一定程度的冗余。因此我们需要首先在算法5.4中压缩 r。

图 5.5 包含流水线级和流控机制的一般性编码器模型

图 5.6　使用削减的输出序列恢复输入

在行1，我们将推导的谓词 $valid(\vec{f})$ 和 $F_{PC}(p,l,r'-1)$ 与在一起。当该公式可满足时，则 r' 是最后一个使得 $F_{PC}(p,l,r') \wedge valid(\vec{f}_{p+l})$ 不可满足的值，我们将其直接直接返回。另一方面，当 $r' \equiv 0$，$F_{PC}(p,l,0)$ 肯定已经在上一个迭代中被测试，且结果为不可满足。此时我们直接返回 0。

这样, 我们从算法5.4得到了一个压缩的 r，使得 $\vec{i}_{p+l}$ 可以被 $<\vec{o}_p,\ldots,\vec{o}_{p+l+r}>$ 唯一决定。

我们进一步要求:

1. 如图5.6所示, l 可以被削减为 0。这意味着 $\vec{i}_p$ 可以被 $<\vec{o}_p,\ldots,\vec{o}_{p+r}>$ 唯一决定。即所有的未来输出。

2. 上述的输出序列 $<\vec{o}_p,\ldots,\vec{o}_{p+r}>$ 能被进一步压缩为 $\vec{o}_{p+r}$。这意味着只需 $\vec{o}_{p+r}$ 即可唯一决定 $\vec{i}_p$。

检验这两个要求等价于检查 $F'_{PC}(p,r) \wedge valid(\vec{f}_{p+l})$ 的不可满足性其中 $F'_{PC}(p,r)$ 定义如下:

$$F'_{PC}(p,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & i_p \equiv 1 \wedge i'_p \equiv 0 \end{array} \right\} \tag{5.13}$$

该要求看起来远远强于 (5.1)。我们将在实验结果中指出他们总能够满足。

5.4.2　推导流水线结构

现在, 基于上述推导的 p 和 r, 我们将公式 (5.13) 中的 F'_{PC} 推广到下面定义的更广泛的形式。它能够检查任意变量 v 在步 j 能否被向量 $\vec{w}$ 在步 k 唯一决定。现在 v 和 $\vec{w}$ 可以使输入，输出或者状态向量。

$$F''_{PC}(p,r,v,j,\vec{w},k) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{w}_k \equiv \vec{w'}_k \\ \wedge & v_j \equiv 1 \wedge v'_j \equiv 0 \end{array} \right\} \tag{5.14}$$

很明显, 当 $F''_{PC}(p,r,v,j,\vec{w},k)$ 不可满足时， $\vec{w}_k$ 能唯一决定 v_j.

对于 $0 \le j \le n-1$, 在第 j 流水线级 $\vec{stg}^j$, 其留空向量 $\vec{f}^j$ 包含所有的能够在第 $j-((n-1)-(p+r))$-th 步被 $\vec{o}$ 在第 $p+r$ 步唯一决定的状态变量 $s \in \vec{s}$。注意在这里不需要约束 $valid(\vec{f_p})$。这可以形式化的定义为:

$$\vec{f}^j := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,j-D,\vec{o},p+r) \\ is\ unsatisfiable \end{array} \right\} \tag{5.15}$$

其中:

$$D \ := \ (n-1)-(p+r) \tag{5.16}$$

而在第 j 流水线级 $\vec{stg}^j$ 中的数据向量 $\vec{d}^j$ 包含能够在第 $j-((n-1)-(p+r))$ 步被 $\vec{o}$ 在第 $p+r$ 步唯一决定的所有 $s \in \vec{s}$。注意这里我们需要强制 $valid(\vec{f_p})$。这可以被形式化的定义为:

$$\vec{d}^j := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,j-D,\vec{o},p+r) \wedge valid(\vec{f_p}) \\ is\ unsatisfiable \end{array} \right\} \tag{5.17}$$

5.5 特征化流水线级和输入的布尔函数

5.5.1 特征化最后一个流水线级的布尔函数

从公式 (5.15) 可知, 每个寄存器 $s \in \vec{f}^{n-1}$ 能过被 $\vec{o}$ 在第 $p+r$ 步唯一决定。也就是, $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ 不可满足且可以划分为:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+r} \equiv 1 \end{array} \right\} \tag{5.18}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & s'_{p+r} \equiv 0 \end{array} \right\} \tag{5.19}$$

因为 $F''_{PC}(p,r,s,p+r,\vec{o},p+r)$ 等价于 $\phi_A \wedge \phi_B$, 所以 $\phi_A \wedge \phi_B$ 不可满足而 ϕ_A 和 ϕ_B 的共同变量集合是 $\vec{o}_{p+r}$。

根据 [13], ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以被计算出来，只引用 $\vec{o}_{p+r}$，并且覆盖所有能使 $s_{p+r} \equiv 1$ 的 $\vec{o}_{p+r}$。同时, $\phi_I \wedge \phi_B$ 不可满足这意味着 ϕ_I 并不覆盖任何使得 $s_{p+r} \equiv 0$ 的 $\vec{o}_{p+r}$。

因此, ϕ_I 可以作为从 $\vec{o}$ 恢复 $s \in \vec{f}^{n-1}$ 的布尔函数。

通过将 $F''_{PC}(p, r, s, p+r, \vec{o}, p+r)$ 替换为 $F''_{PC}(p, r, s, p+r, \vec{o}, p+r) \wedge valid(f_p)$，我们可以类似的特征化恢复 $s \in \vec{d}^{n-1}$ 的布尔函数。

5.5.2 特征化恢复其他流水线级的布尔函数

根据图5.5, $\vec{f}^{j}$ 在第 $j-D$ 步可以被 $\vec{stg}^{j+1}$ 在第 $j-D+1$ 步唯一决定。因此我们将不可满足公式 $F''_{PC}(p, r, s, j-D, \vec{stg}^{j+1}, j-D+1)$ 划分为下列两个公式：

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & s_{j-D} \equiv 1 \end{array} \right\} \tag{5.20}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{j+1}_{j-D+1} \equiv \vec{stg'}^{j+1}_{j-D+1} \\ \wedge & s'_{j-D} \equiv 0 \end{array} \right\} \tag{5.21}$$

再一次, ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以被构造出来, 并用做从 $\vec{stg}^{j+1}$ 恢复 $s \in \vec{f}^{j}$ 的布尔函数。

类似的, 将 $F''_{PC}(p, r, s, j-D, \vec{stg}^{j+1}, j-D+1)$ 替换为 $F''_{PC}(p, r, s, j-D, \vec{stg}^{j+1}, j-D+1) \wedge valid(f_p)$，我们能特征化从 $\vec{stg}^{j+1}$ 恢复 $s \in \vec{d}^{j}$ 的布尔函数。

5.5.3 特征化从第 0 级流水线恢复输入向量的布尔函数

根据图5.5, $\vec{f}$ 在第 p 步能够被 $\vec{stg}^{0}$ 在第 p 步唯一决定。$F''_{PC}(p, r, i, p, \vec{stg}^{0}, p)$ 不可满足并可以划分为以下两个公式：

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & i_p \equiv 1 \end{array} \right\} \tag{5.22}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{0}_{p} \equiv \vec{stg'}^{0}_{p} \\ \wedge & i'_p \equiv 0 \end{array} \right\} \tag{5.23}$$

表 5.1 Benchmark 和实验结果

Names	编码器				[3] 产生的解码器			本文产生的解码器		
	# in/out	# reg	面积	解码器藐视	运行时间	延迟 (ns)	面积	运行时间	延迟 (ns)	面积
pcie	10/11	23	326	PCIE 2.0 [28]	0.37	7.20	624	8.08	5.89	652
xgxs	10/10	16	453	以太网 clause 48 [27]	0.21	7.02	540	4.25	5.93	829
t2eth	14/14	49	2252	以太网 clause 36 [27]	12.7	6.54	434	430.4	6.12	877
scrambler	64/64	58	1034	inserting 01 flipping	no pipeline stages found					
xfi	72/66	72	7772	以太网 clause 49 [27]						

再一次, ϕ_A 相对于 ϕ_B 的 Craig 插值 ϕ_I 可以被用作从 $\vec{stg}^0$ 恢复 $i \in \vec{f}$ 的布尔函数。

类似的, 通过替换 $F''_{PC}(p,r,i,p,\vec{stg}^0,p)$ 为 $F''_{PC}(p,r,i,p,\vec{stg}^0,p) \wedge valid(\vec{f_p})$，我们可以特征化从 $\vec{stg}^0$ 恢复 $i \in \vec{d}$ 的布尔函数。

5.6 实验结果

我们使用 OCaml 语言实现了上述算法，并使用 MiniSat 1.14 [17] 求解产生的 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 内存, 和 CentOS 5.4 Linux 操作系统的服务器上。

5.6.1 比较时间和面积

表5.1 给出本文中使用的 benchmark。第 2 和 3 分别给出输入, 输出和寄存器个数。第 4 列给出了将编码器映射至 LSI10K 库所得到的面积。本文中, 所有的面积和延迟使用同样的设置得到。

第 6 到 8 列分别给出了论文 [3] 的算法产生非流水解码器的运行时间，以及该解码器的延迟和面积。而第 9 到 11 列分别给出了本文算法的类似信息。

比较第 7 和 10 列可知延迟得到了明显的改善。

一个比较令人惊讶的事实是，两个最大的 benchmarks scrambler 和 xfi 并不包含流水线。我们研究并确认了这一点。他们的面积如此之大是因为使用了很宽的 64 到 72 位数据路径。

5.6.2 PCIE 推导的流水线结构

对于 benchmark pcie，存在两个流水线级, 其中包含的流控向量和数据向量如图5.2所示。

表 5.2　pcie 推导的流水线结构

	input	pipeline stage 0	pipeline stage 1
流控向量	CNTL_TXEnable_P0	InputDataEnable_P0_reg	OutputData_P0_reg[9:0] OutputElecIdle_P0_reg
流控谓词	CNTL_TXEnable_P0	InputDataEnable_P0_reg	true
数据向量	TXDATA[7:0] TXDATAK	InputData_P0_reg[7:0] InputDataK_P0_reg	

有一个有趣的事实是流水线级 1 的数据向量是空集。而所有的流水线寄存器都被识别成为流控向量。我们研究了源代码，发现这些流水线寄存器全部都被直接送给输出向量。因此他们很明显都能够被 ∂ 唯一决定。伊霓裳这并不影响所产生的解码器的正确性。

5.6.3　xgxs 推导的流水线结构

对于 benchmark xgxs, 只有一级流水线，其中的流控和数据向量如图5.3所示。

表 5.3　xgxs 推导的流水线结构

	input	pipeline stage 0
流控向量	bad_code	bad_code_reg_reg
流控谓词	!bad_code	!bad_code_reg_reg
数据向量	encode_data_in[7:0] konstant	ip_data_latch_reg[2:0] plus34_latch_reg data_out_latch_reg[5:0] konstant_latch_reg kx_latch_reg minus34b_latch_reg

表 5.4　t2ether 推导的流水线结构

	input	pipeline stage 0	pipeline stage 1	pipeline stage 2	pipeline stage 3
流控向量	tx_enc_ctrl_sel[3:0]	qout_reg_0_8 qout_reg_2_4 qout_reg_1_4	qout_reg_0_9 qout_reg_1_5 qout_reg_2_5 qout_reg_0_10	qout_reg[9:0]_2	qout_reg[7:1]_3 qout_reg_8_1 qout_reg_9_1 qout_reg_3_4 qout_reg_0_4 qout_reg_3_5 qout_reg_0_7 sync1_reg1 sync1_reg Q_reg1 Q_reg
数据向量	txd[7:0]	qout_reg[7:0]	qout_reg[7:0]_1		

5.6.4　t2ether 推导的流水线结构

对于 benchmark t2ether, 有 3 级流水线，如图5.4所示。流控谓词比较复杂，因此我们将他们单独列在下面而不是图5.4中。输入流控谓词 f 为:

$$
\begin{aligned}
&(tx\_enc\_ctrl\_sel[2] \,\&\, tx\_enc\_ctrl\_sel[3])| \\
&(tx\_enc\_ctrl\_sel[2] \,\&\, !tx\_enc\_ctrl\_sel[3] \,\&\, !tx\_enc\_ctrl\_sel[0] \,\&\, tx\_enc\_ctrl\_sel[1])| \\
&(!tx\_enc\_ctrl\_sel[2] \,\&\, tx\_enc\_ctrl\_sel[3])| \\
&(!tx\_enc\_ctrl\_sel[2] \,\&\, !tx\_enc\_ctrl\_sel[3] \,\&\, tx\_enc\_ctrl\_sel[0])
\end{aligned}
\tag{5.24}
$$

第零级流控谓词 $valid(\vec{f}^0)$:

$$
\begin{aligned}
&(qout\_reg\_2\_4 \,\&\, qout\_reg\_1\_4 \,\&\, !qout\_reg\_0\_8)| \\
&(!qout\_reg\_2\_4 \,\&\, qout\_reg\_0\_8)
\end{aligned}
\tag{5.25}
$$

第一级流控谓词 $valid(\vec{f}^1)$ 为:

$$
\begin{aligned}
&(qout\_reg\_2\_5 \ \&\ qout\_reg\_1\_5 \ \&\ qout\_reg\_0\_10 \ \&\ !qout\_reg\_0\_9)| \\
&(qout\_reg\_2\_5 \ \&\ qout\_reg\_1\_5 \ \&\ !qout\_reg\_0\_10)| \\
&(qout\_reg\_2\_5 \ \&\ !qout\_reg\_1\_5 \ \&\ !qout\_reg\_0\_10)| \\
&(!qout\_reg\_2\_5 \ \&\ qout\_reg\_0\_10 \ \&\ qout\_reg\_0\_9)| \\
&(!qout\_reg\_2\_5 \ \&\ !qout\_reg\_0\_10)
\end{aligned}
\tag{5.26}
$$

最后两级的流控谓词 $valid(\vec{f}^2)$ 和 $valid(\vec{f}^3)$ 均为 *true*。

5.7 相关工作

5.7.1 对偶综合

第一个对偶综合算法由沈 [1] 提出。他通过迭代的展开迁移函数来检查解码器的存在性，并通过遍历所有的输出向量取值来特征化解码器的布尔函数。其主要弱点在于该算法不停机，而且特征化解码器函数时速度太慢。

沈 et al.[3] 和 Liu et al.[6] 通过在状态序列上搜索环来解决停机问题。而 [5, 6] 通过 Craig 插值 [11] 高效的特征化解码器函数。

沈 et al.[5] 自动推导使得解码器存在的配置断言。

秦 et al. [12] 提出了第一个能处理流控机制的对偶综合算法。

Tu 和 Jiang [8] 提出了一个突破性的算法以在恢复编码器输入时考虑其非限界的历史。

5.7.2 程序取反

根据 Gulwani [31], 程序取反意味着为特定程序 P 生成一个具有反功能的程序 P^{-1}。因此, 程序取反非常类似于我们的对偶综合。

最早的程序取反算法基于 proof 的算法 [32], 只能处理非常小的程序和非常简单的语法。

Glück et al. [33] 取反一阶函数语言的程序，通过基于 LR 的算法来去除非确定性。. 然而使用函数语言使得其应用场景和我们有很大的差别。

Srivastava et al. [34**?**] 假设反程序和原始程序的结构是类似的, 因此可能的反程序结构可以通过挖掘原始程序中的谓词，表达式和控制流得到。这和本文发掘流水线级和流控机制类似。该算法迭代的去掉不满足要求的解空间直至剩下正确的解。

5.8 结论

本文提出了第一个能同时处理流控机制和流水线的对偶综合算法。实验结果表明本文算法总能够正确的产生带有流控机制和流水线的解码器。

第六章　结束语

本章对全文进行总结，并对进一步研究工作进行展望。

6.1　工作总结

拓扑压缩是无线传感器网络研究中的重要问题。本文以提高方法的可用性和效能为研究目标，以保证较低的几何失真率为贯穿始终的标准，系统地研究了拓扑压缩技术中的一些重要问题。具体而言，本文主要对以下几个重要问题进行了深入研究。

第一，不依赖位置信息的拓扑骨干提取问题。拓扑骨干提取是拓扑压缩的重要问题。目前已有的不依赖位置的拓扑骨干提取算法大部分依赖特殊的网络假设，或无法提取出确定性的、严格符合实际网络形状的拓扑骨干。本文针对已有方法中的局限性，提出了一种仅依赖局部连通性信息，具有良好鲁棒性的拓扑骨干提取算法。算法利用了仅依赖局部连通性信息的基于 MDS 的边界识别算法，提出了骨干带网络构建方法以及高效的图变换工具 HPT，并设计了一种灵活有效的骨干叶节点判定方法。本文通过理论证明以及大量的仿真实验验证了算法的有效性和性能，实验结果显示算法能够有效地适用于具有各种不同形状的网络，提取出具有良好连通性和形状的拓扑骨干，且对多种关键的网络参数具有良好的鲁棒性。

第二，不依赖位置信息的虫洞拓扑检测问题。虫洞攻击是无线自组织与传感器网络中一种严重的攻击。现有的大部分虫洞检测方法严格依赖于特殊的硬件设备或理想的网络假设，从而在很大程度上限制了这些方法的可用性。而现有的基于网络连通性的检测方法都是基于利用离散域的局部的虫洞特征，或者连续域的全局的网络特征。针对现有方法的局限性，本文深入挖掘虫洞攻击对全局的网络拓扑造成的本质影响，首次提出了一种仅依赖局部连通性信息且能够直接从离散域捕获虫洞造成的全局拓扑症状的虫洞检测方法，称为 WormPlanar。WormPlanar 巧妙地利用了虫洞攻击对网络平面化造成的影响，能够有效地检测和定位不同网络条件下的虫洞攻击。本文从理论上充分地证明了 WormPlanar 方法的正确性，并通过大量的仿真实验验证了算法的有效性和性能。

第三，路由路径记录问题。路由路径记录是无线传感器网络中重要的功能，对于改善网络状态的可见性以及提供细粒度的网络管理具有重要的作用。目前的相关研究均无法获得网络中每个数据包的完整路径信息。本文首次正式地提出并系统地研究无线传感器网络的路由路径记录问题，设计了一种轻量级的、在实际的大规模网络中可用的路由路径压缩和恢复方法，称为 PathZip。PathZip 巧妙地

设计了基于哈希的路径压缩和恢复机制，将大部分的计算和存储开销从传感器节点转移至基站。同时，本文还设计了分别基于拓扑和基于几何的技术，有效地降低路径恢复的开销。本文通过理论分析和大量的仿真实验验证 PathZip 方法的有效性和性能，实验结果证明 PathZip 能够在较低的计算和存储开销的基础上，实时地记录网络中每个数据包的精确传输路径。

第四，不精确位置信息下的贪婪地理路由。贪婪地理路由由于其简单高效性在无线传感器网络中得到了广泛的研究和应用。为了设计在实际的大规模网络系统中可用的贪婪地理路由协议，研究者进行了大量的工作，特别是针对局部最小问题上提出了大量的解决方案。之前的各种方法具有各自的优势和适用范围，在一定的网络假设条件下有效地克服了局部最小问题。本文结合之前的各类方法的优势，提出了一种细粒度的层次式贪婪地理路由方法，称为 FLYER。FLYER 不依赖于精确的位置信息或全局的状态信息，在节点位置误差率不超过一定上限值时具有传输保证。FLYER 方法以完全分布式的方式运行，计算和存储开销均非常低，且能够输出具有较低的失真率以及良好的负载均衡性能的路由路径。本文通过理论分析和大量的仿真实验验证了 FLYER 的有效性和性能，证明了 FLYER 在多项性能指标上相对于之前的方法具有明显的优势。

6.2 研究展望

本文深入研究了无线传感器网络拓扑压缩技术中几个重要的问题。虽然本文在拓扑压缩问题中取得了一定的研究成果，但由于基于连通性的拓扑结构研究问题的挑战性，该领域还存在许多问题需要进一步的研究。在本文研究的基础上，需要进一步研究的课题包括：

第一，轻量级的局部化拓扑骨干识别算法设计。本文设计的骨干提取算法虽然以分布式的方式实现了鲁棒的骨干提取，但算法中一些步骤涉及到比较复杂的分布式协作过程，在特殊网络情况下可能需要范围较大的消息交换。因此，如何进一步地深入挖掘和利用网络的拓扑结构，设计仅依赖严格的局部连通性信息的骨干提取算法，或是在算法复杂度和拓扑骨干质量之间建立一定的折衷，是一项值得研究且具有一定挑战性的工作。

第二，鲁棒的虫洞检测算法设计。本文设计的虫洞检测算法依赖于对节点的邻居子图的平面化特征的判断。当网络密度非常低时，在仅利用局部连通性信息的情况下可能无法捕获网络子图的非平面化特征，从而导致算法无法成功检测出所有的虫洞节点。因此，如何利用局部连通性信息对网络的非平面化特征进行更可靠的描述，提高算法在网络密度非常低情况下的检测率，是未来改进算法值得研究的问题。

第三，轻量级的理由路径恢复机制研究。本文设计的路由路径记录方法在一定程度上利用了路由路径之间的时间相关性和空间相关性，但是在网络路由的动态性较强时，路径恢复过程仍将涉及比较高的计算开销。如何进一步地通过挖掘路由路径之间的相关性，设计轻量级的路径恢复机制，是未来改进算法值得研究的问题。另外，在虚拟路径机制中，如何设计虚拟路径算法使得近似区域内的候选节点的数量最小化，也是一个值得研究的问题。

第四，动态网络中的层次式贪婪地理路由设计。在本文设计的路由算法中，需要利用拓扑平面化算法以构建平面化地标网络。在网络动态性较强，或者存在一定数量的移动节点时，如何保证平面化算法的效率和实时性，是一项值得研究且具有一定挑战性的工作。从另一个角度来看，如何利用具有移动能力的锚节点来辅助贪婪地理路由方法的设计，也是一个有价值的研究方向。

致　谢

值此成文之际，谨向在我攻读博士期间给予我指导、关心、支持和帮助的老师、领导、同学和亲人们致以衷心的感谢！

首先，衷心感谢我的导师廖湘科老师！从 06 年进入科大读取硕士开始我就有幸成为了廖老师的学生，在多年来的学习和生活中得到了廖老师的悉心指导和无私关怀。在课题选择和问题解决过程中，您以敏锐的学术洞察力和深厚的科研经验，高屋建瓴地为我论证把关。您在百忙之中仍抽出时间对我的课题进行指导，及时为我解决困难和提供帮助。与您的讨论使我能够抓住课题中的关键问题，明确研究目标。没有您的指导和帮助，我的研究工作将不可能顺利完成。廖老师对我的言传身教将使我终身受益，您严谨的治学作风、高深的学术造诣、忘我的工作态度和勇攀高峰的精神将永远影响和激励着我。感谢我的师母谢志军女士，您在生活中给与了我和整个师门的同学们无微不至的关怀，您的热情大方、细心体贴和乐观开朗时刻温暖和鼓励着我们每个学生，使我们组成紧密团结的大家庭，共同成长。

衷心感谢董德尊老师！在整个博士课题研究过程中，您始终给与我悉心指导和无私帮助，使我的研究工作能够顺利进行。与您的每次讨论都能够开阔我的研究思路，丰富我的理论知识；您严谨的工作态度、深厚的理论功底、创新的研究思路，无不使我深感敬佩并始终将您作为我学习的榜样。您在繁忙的工作中仍抽出时间与我讨论、帮我修改论文。仍记得我的第一篇论文在您修改之后的面目全非，也记得数次论文被拒之后您微笑着的鼓励并耐心地与我讨论修改方案。在生活中您是我的好师兄，给与了我很多的鼓励和帮助。您作为我的老师、师兄，同时又是我真诚的朋友！在此，向董老师致以我诚挚的敬意和衷心的感谢！

衷心感谢卢宇彤老师给与我的耐心指导和无私帮助。卢老师深厚的学术功底、丰富的科研经验、热情爽朗的性格、和蔼可亲的待人方式，都使我非常的敬佩。感谢曾经给与过我指导和帮助的曹宏嘉老师、周恩强老师、谢闵老师、陈海涛老师、蒋艳凰老师，你们勤奋投入的工作态度和超强的科研与工程能力都是我学习的目标。

衷心感谢李姗姗老师，您在学习和生活上给与了我很多的帮助和鼓励，在繁忙的工作中组织师门召开组会，使我们能够凝聚在一起热烈地交流和讨论，共同进步。感谢杨沙洲老师、彭绍亮老师、付松龄老师、黄辰林老师、王蕾老师、王

小平老师。你们一直密切关注我的课题进展情况，给与了我很多的指导和帮助，对我的论文提出了大量宝贵的意见和建议。

感谢吴庆波老师、戴华东老师、何连跃老师、陈松政老师、邵立松老师、丁澉老师。你们在我工程实践和课题研究过程中都提出了良好的建议。

感谢师门的谢欣伟、熊伟、付强、刘晓东、郑思、林彬、张菁、黄詠、郭勇、朱浩、任静、范小康、申彤、张峰、雷斐、崔英博、李存禄、徐尔茨、贾周阳、柴燕涛。你们在几年时间里给与了我很多的帮助，使我时刻感受到师门大家庭的温暖。我们大家一起在实验室埋头苦读、在游乐场放声欢笑、在足球场奋力拼搏的日子，将成为我永远的美好回忆。

感谢曾与我在同一个实验室奋斗的刘德峰、彭林、贾建斌、毛华坚、倪时策、胡维、吴强、沈洁、牛海波、郭鹏宇、刘金磊、王春光、马文琪、王文竹等同学，我们在学习上频繁地交流，在生活上互相帮助，使我的博士生活充满了乐趣。

感谢几年来朝夕相处的罗阳、荀长庆、徐新海、王耀华、曾瑶源、李柱、游皓聃、黄冕、韦兴军、胥清化、袁建国、高航、王勇、曹维、黄传洪、徐林等同学。感谢五队 06 级和七队 09 级的所有同学，我们一起经历了科大的学习和生活，并留下了许多美好的回忆。

感谢学院、学员大队、学员队各级领导对我的教育、关心和帮助，你们的辛勤工作为我们创造了良好的学习和生活环境。

特别感谢含辛茹苦将我抚育成人的父母亲，对你们的感激之情无法用语言来表达。在我多年的求学之路上，你们始终给与我全身心的支持和关爱。我无法经常陪伴在你们身边，唯有在将来的工作中刻苦努力、做出更大的成绩，以报答你们的养育之恩。祝愿你们永远健康幸福!

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, Zhang J. Inferring assertion for complementary synthesis [C/OL]. In Proceedings of the 2011 International Conference on Computer-Aided Design. San Jose, CA, USA, 2011: 404–411. http://dx.doi.org/10.1109/ICCAD.2011.6105361.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] Qin Y, Shen S, Wu Q, et al. Complementary Synthesis for Encoder with Flow Control Mechanism [J]. accepted by ACM Transactions on Design Automation of Electronic Systems.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Clarke E M, Grumberg O, Jha S, et al. Counterexample-Guided Abstraction Refinement [C/OL] // Emerson E A, Sistla A P. In Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings. 2000: 154–169. http://dx.doi.org/10.1007/10722167_15.

[23] Yuan J, Albin K, Aziz A, et al. Constraint synthesis for environment modeling in functional verification [C/OL]. In Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003. 2003: 296–299. http://doi.acm.org/10.1145/775832.775909.

[24] Amla N, McMillan K L. A Hybrid of Counterexample-Based and Proof-Based Abstraction [C/OL] // Hu A J, Martin A K. In Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings. 2004: 260–274. http://dx.doi.org/10.1007/978-3-540-30494-4_19.

[25] McMillan K L. Interpolation and SAT-Based Model Checking [C/OL] // Jr W A H, Somenzi F. In Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings. 2003: 1–13. http://dx.doi.org/10.1007/978-3-540-45069-6_1.

[26] McMillan K L. An interpolating theorem prover [J/OL]. Theor. Comput. Sci. 2005, 345 (1): 101–121. http://dx.doi.org/10.1016/j.tcs.2005.07.003.

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

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

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

[30] 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.

[31] 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.

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

[50] 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.

[51] 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).

[52] 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.