第一章　面向流控机制和流水线的对偶综合

1.1　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 [1–6] 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 [7] 不能满足该要求。如图1.1a) 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1.1b) 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 $\vec{d}$。

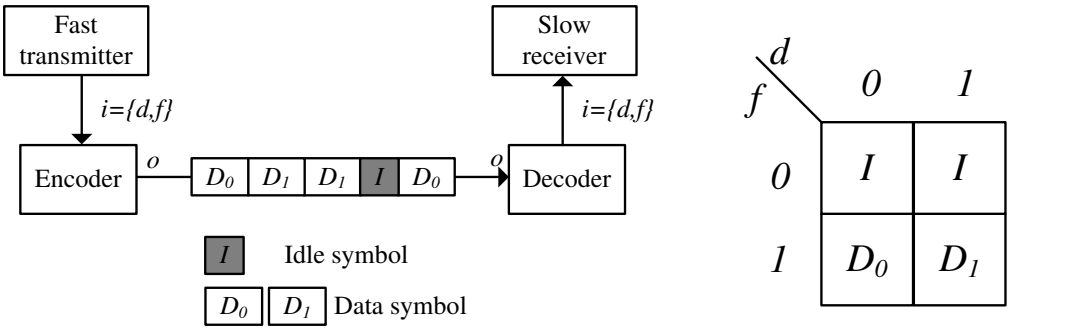
秦 et al. [8] 首次提出了能够处理流控机制的对偶综合算法。该算法首先找到所有的能够被 $\vec{o}$ 唯一决定的 $i \in \vec{f}$。然后推导一个能使得 $\vec{d}$ 被 $\vec{o}$ 唯一决定的谓词 $valid(\vec{f})$。

同时，如图1.2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 [8] 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. [8] 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. [9] 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1.2 介绍相关的背景知识；小节1.3 介绍本文算法的整体结构；小节1.4 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1.5 为 $s\vec{t}g^j$ 和 $\vec{i}$ 特征化布尔函数。小节1.6 和1.7 分别给出实验结果和相关工作。最后小节1.8 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 1.1　带有流控机制的编码器

1.2　背景知识

1.2.1　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。

1.2.2　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。

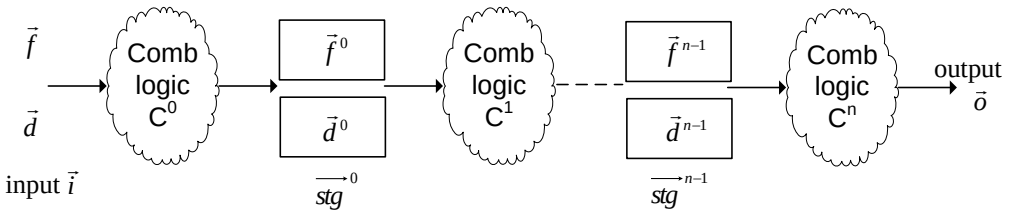


图 1.2　带有流水线和流控机制的编码器

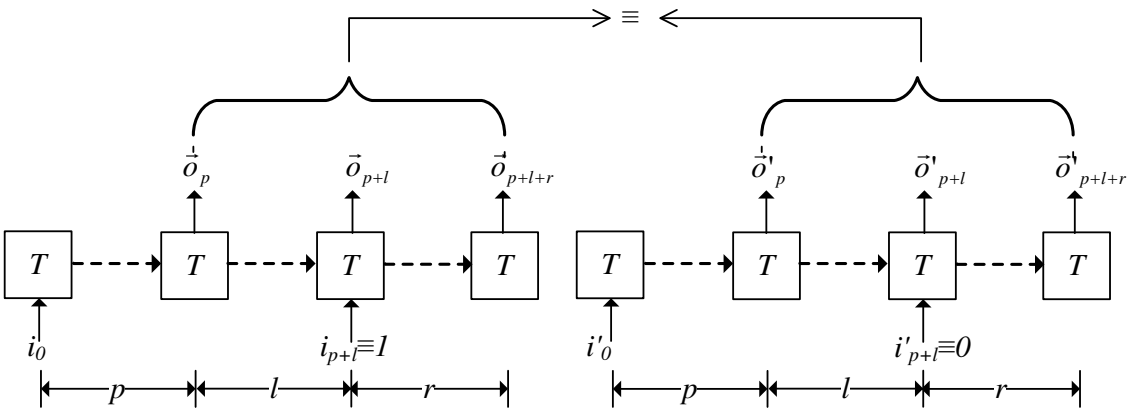


图 1.3　sound 和 complete 方法

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 $\vec{s}_n \equiv \vec{s}_m$。

1.2.3　寻找 $\vec{f}$ 的停机算法

秦 et al. [8] 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。.

1.2.3.1　sound 算法

如图1.3a) 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 (1.1) 中的 $F_{PC}(p, l, r)$ 的不可满足性。行 1 对应于图1.3a) 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 (1.1) 不可满足 i 肯定是 $\vec{f}$ 的成员。

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

1.2.3.2　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 1.1 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1.3b) 所示的方法去检查第二种情形。该方法类似于1.3a)，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 (1.2) 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 $i \notin \vec{f}$。

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

1.2.3.3 通过算法1.1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 [8] 。

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

该算法也是由秦 et al. [8] 提出的。它首先给出算法1.2，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1.4所

算法 1.2 *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1.2 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 $valid(\vec{f})$。

1.2.4.1 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1.2 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 $FSAT_R(\vec{a})$。

1.2.4.2 计算 $valid(\vec{f})$ 的单调递增下估计

通过将公式 (1.1) 中的 i 替换为算法1.1中推导的 $\vec{d}$，我们有:

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

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 (1.3) 的第三行，我们定义 $T_{PC}(p,l,r)$:

$$T_{PC}(p,l,r) := \left\{ \bigwedge_{m=p}^{p+l+r}\vec{o}_m \equiv \vec{o'}_m \right\} \tag{1.4}$$

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

$$F'^{d}_{PC}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & t \equiv T_{PC}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \end{array} \right\} \tag{1.5}$$

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：

$$\vec{a} := \vec{f}_{p+l} \tag{1.6}$$

$$\vec{b} := \vec{d}_{p+l} \cup \vec{d'}_{p+l} \cup \vec{s}_0 \cup \vec{s'}_0 \cup \bigcup_{0\leq x\leq p+l+r, x\neq(p+l)} (\vec{i}_x \cup \vec{i'}_x) \tag{1.7}$$

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：

$$FSAT_{PC}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{PC}(p,l,r,t),\vec{a},\vec{b},t) \tag{1.8}$$

如图1.4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。

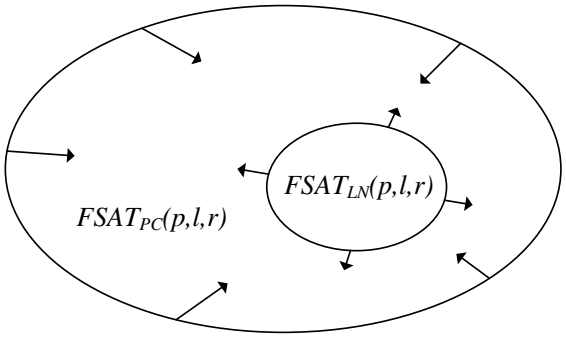


图 1.4　The monotonicity of $FSAT_{PC}(p,l,r)$ and $FSAT_{LN}(p,l,r)$

算法 1.3 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** $\neg FSAT_{LN}(p,l,r)$

1.2.4.3 计算 $valid(\vec{f})$ 的单调递减上估计

类似的, 我们定义:

$$T_{LN}(p,l,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=p}^{p+l+r} \vec{o}_m \equiv \vec{o'}_m \\ \wedge & \bigvee_{x=0}^{p-1} \bigvee_{y=x+1}^{p} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+1}^{p+l-1} \bigvee_{y=x+1}^{p+l} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \\ \wedge & \bigvee_{x=p+l+1}^{p+l+r-1} \bigvee_{y=x+1}^{p+l+r} \{\vec{s}_x \equiv \vec{s}_y \wedge \vec{s'}_x \equiv \vec{s'}_y\} \end{array} \right\} \tag{1.9}$$

$$F'^{d}_{LN}(p,l,r,t) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+l+r} \{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+l+r} \{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & t \equiv T_{LN}(p,l,r) \\ \wedge & \vec{d}_{p+l} \neq \vec{d'}_{p+l} \end{array} \right\} \tag{1.10}$$

$$FSAT_{LN}(p,l,r) := CharacterizingFormulaSAT(F'^{d}_{LN}(p,l,r,t), \vec{a}, \vec{b}, t) \tag{1.11}$$

如图1.4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。

1.2.4.4 使用算法1.3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 [8] 。

1.3 算法框架

1.3.1 编码器的一般性模型

如图1.5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 1.4 压缩 r

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

和 [8] 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：

$$\begin{aligned} \vec{stg}^{\,0} &:= C^0(\vec{i}) \\ \vec{stg}^{\,j} &:= C^j(\vec{stg}^{\,j-1}) \quad 1 \le j \le n-1 \\ \vec{o} &:= C^n(\vec{stg}^{\,n-1}) \end{aligned} \tag{1.12}$$

在本文中，上标始终意味着流水线级, 而下标，, 如小节1.2.2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。

1.3.2 算法框架

基于图1.5所示的编码器结构，我们算法的框架为：

1. 调用算法1.1 已将 $\vec{i}$ 划分为 $\vec{f}$ 和 $\vec{d}$。
2. 调用算法1.3 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1.4, 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1.5, 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。

1.4 推导流水线结构

1.4.1 压缩 r 和 l

由于算法1.3 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1.4中压缩 r。

图 1.5 包含流水线级和流控机制的一般性编码器模型

图 1.6　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1.4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1.6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:

$$F'_{PC}(p,r) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & i_p \equiv 1 \wedge i'_p \equiv 0 \end{array} \right\} \tag{1.13}$$

该要求看起来远远强于 (1.1)。我们将在实验结果中指出他们总能够满足。

1.4.2　推导流水线结构

现在, 基于上述推导的 p 和 r, 我们将公式 (1.13) 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。

$$F''_{PC}(p,r,v,j,\vec{w},k) := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{w}_k \equiv \vec{w'}_k \\ \wedge & v_j \equiv 1 \wedge v'_j \equiv 0 \end{array} \right\} \tag{1.14}$$

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 $s\vec{t}g^j$, 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:

$$\vec{f}^j := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,j-D,\vec{o},p+r) \\ is\ unsatisfiable \end{array} \right\} \tag{1.15}$$

其中:

$$D \ := \ (n-1)-(p+r) \tag{1.16}$$

而在第 j 流水线级 $s\vec{t}g^j$ 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:

$$\vec{d}^j := \left\{ s \in \vec{s} \mid \begin{array}{c} F''_{PC}(p,r,s,j-D,\vec{o},p+r) \wedge valid(\vec{f_p}) \\ is\ unsatisfiable \end{array} \right\} \tag{1.17}$$

1.5 特征化流水线级和输入的布尔函数

1.5.1 特征化最后一个流水线级的布尔函数

从公式 (1.15) 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:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1},\vec{o}_m) \equiv T(\vec{s}_m,\vec{i}_m)\} \\ \wedge & s_{p+r} \equiv 1 \end{array} \right\} \tag{1.18}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1},\vec{o'}_m) \equiv T(\vec{s'}_m,\vec{i'}_m)\} \\ \wedge & \vec{o}_{p+r} \equiv \vec{o'}_{p+r} \\ \wedge & s'_{p+r} \equiv 0 \end{array} \right\} \tag{1.19}$$

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 [9], 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。

1.5.2 特征化恢复其他流水线级的布尔函数

根据图1.5, $\vec{f}^j$ 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:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & s_{j-D} \equiv 1 \end{array} \right\} \tag{1.20}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^{j+1}_{j-D+1} \equiv \vec{stg'}^{j+1}_{j-D+1} \\ \wedge & s'_{j-D} \equiv 0 \end{array} \right\} \tag{1.21}$$

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。

1.5.3 特征化从第 0 级流水线恢复输入向量的布尔函数

根据图1.5, $\vec{f}$ 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:

$$\phi_A := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s}_{m+1}, \vec{o}_m) \equiv T(\vec{s}_m, \vec{i}_m)\} \\ \wedge & i_p \equiv 1 \end{array} \right\} \tag{1.22}$$

$$\phi_B := \left\{ \begin{array}{cc} & \bigwedge_{m=0}^{p+r}\{(\vec{s'}_{m+1}, \vec{o'}_m) \equiv T(\vec{s'}_m, \vec{i'}_m)\} \\ \wedge & \vec{stg}^0_p \equiv \vec{stg'}^0_p \\ \wedge & i'_p \equiv 0 \end{array} \right\} \tag{1.23}$$

表 1.1　Benchmark 和实验结果

Names	编码器				[2] 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 [13]	0.37	7.20	624	8.08	5.89	652
xgxs	10/10	16	453	以太网 clause 48 [14]	0.21	7.02	540	4.25	5.93	829
t2eth	14/14	49	2252	以太网 clause 36 [14]	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 [14]						

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。

1.6　实验结果

我们使用 OCaml 语言实现了上述算法，并使用 MiniSat 1.14 [12] 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。

1.6.1　比较时间和面积

表1.1 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 [2] 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。

1.6.2　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1.2所示。

表 1.2　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 $\vec{o}$ 唯一决定。伊霓裳这并不影响所产生的解码器的正确性。

1.6.3　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1.3所示。

1.6.4　t2ether 推导的流水线结构

对于 benchmark t2ether, 有 3 级流水线，如图1.4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1.4中。输入流控谓词 f 为:

表 1.3　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 1.4　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		

(*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])

(1.24)

第零级流控谓词 $valid(\vec{f}^0)$：

(*qout_reg*_2_4 & *qout_reg*_1_4 & !*qout_reg*_0_8)|
(!*qout_reg*_2_4 & *qout_reg*_0_8)　(1.25)

第一级流控谓词 $valid(\vec{f}^1)$ 为：

(*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)|　(1.26)
(!*qout_reg*_2_5 & *qout_reg*_0_10 & *qout_reg*_0_9)|
(!*qout_reg*_2_5 & !*qout_reg*_0_10)

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

1.7 相关工作

1.7.1 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.[2] 和 Liu et al.[4] 通过在状态序列上搜索环来解决停机问题。而 [3, 4] 通过 Craig 插值 [11] 高效的特征化解码器函数。

沈 et al.[3] 自动推导使得解码器存在的配置断言。

秦 et al. [8] 提出了第一个能处理流控机制的对偶综合算法。

Tu 和 Jiang [6] 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。

1.7.2 程序取反

根据 Gulwani [15], 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 [16], 只能处理非常小的程序和非常简单的语法。

Glück et al. [17] 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. [18**?**] 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。

1.8 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。

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, 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[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] 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.

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

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

[15] 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.

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

[17] 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.

[18] 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.