Control
and Cybernetics
VOL. 21 (1992) No. 3/4

Substitution algorithm for binary chains[1]

by

Jacek Gawin and Paweł Fochtman

Department of Technology,
Division of Electronics and Computer Science,
Silesian University,
ul. Będzińska 60
41–200 Sosnowiec, Poland

The paper presents an algorithm for the Binary Chains algebra, used for functional description of digital circuits in automatic test pattern generation. An example of its performance and a comparison with other algorithms are included.

1. Introduction

The Binary Chains as an economical form of the Binary Decision Diagrams [1, 2, 7] are one of the boolean function notations. The inventors of Binary Chains are K. Sapiecha and T. Czichon [6]. The formal definition of the Binary Chains has been presented in [3]. They are suited for an easy calculation of boolean expressions on complicated functions. Besides, they can be calculated for the variables of a digital circuit from its procedural description in any Hardware Description Language [4, 5]. Therefore the Binary Chains algebra was used for

[1]Sponsored by KBN project no. 8 0275 91 01

the test generation method [6]. Some algorithms of this algebra are presented in [3].

The substitution of the Binary Chains describing some variables so as to form other Binary Chains which contain these variables are frequently used during test generation calculations by the Automatic Test Equipment [7]. There are no publications on the algorithms of such substitution. The present paper describes three original substitution algorithms and discusses them.

Section 2 of this paper outlines the problem. In Sections 3, 4 and 5 the consecutive versions of the algorithm looked for are presented. An example of performance of the last, recurrent algorithm is shown in Section 6. Its qualities are specified in Section 7.

2. Notation and problem formulation

The following notation shall be used throughout the paper:

α – a logic variable;
X – a Binary Chain containing variable α;
$\mathcal{A}$ – a Binary Chain describing variable α as a function;
$\mathcal{X}$ – the sought Binary Chain, equivalent to X.

We have to find the Binary Chain $\mathcal{X}$ which is equal to X, but in which instead of the variable α, the Binary Chain $\mathcal{A}$ appears (see Example 1).

Example 1

$$\begin{aligned} &\alpha = b \\ &X = a, b, 0, 1, b, 1, 0 \quad (X = a \cdot \overline{b} + \overline{a} \cdot b); \\ &\mathcal{A} = c, 0, d, 1, 0 \qquad (b = \overline{c} \cdot d). \end{aligned}$$

The solution sought is:

$$\mathcal{X} = a, c, 1, d, 0, 1, c, 0, d, 1, 0 \quad (X = a \cdot c + a \cdot \overline{d} + \overline{a} \cdot \overline{c} \cdot d)$$

The chain $\mathcal{X}$ should be reduced and the variables appearing in it should occur according to specified (ascending) order. Noncompliance with the first condition will cause quick increase of computational complexity. If the second condition is not satisfied then subsequent calculations may be erroneous due to variable looping (see Examples 2 and 3).

3. Algorithm 1

```
procedure Algorithm1(X, α, A, X);
begin
  X:='';
  sx:=sizeOf(X);
  sa:=sizeOf(A);
  for ix:=1 to sx do
  if X[ix]=α then
    for ia:=1 to sa do
    if A[ia] = const then
      X:=X+subchain(X,ix,const))
    else                    {A[ia] is variable}
      X:=X+A[ia]
   else
     X:=X+X[ix];
 end;
```

The proposed algorithm is simple and rapid. It turns out, however, that variables in the resulting chain are not ordered (see Example 2).

Example 2

$\alpha = a;$
$X = a, b, 1, 0, 0; \quad (X = a \cdot b);$
$\mathcal{A} = c, 1, d, 1, 0; \quad (a = c + d).$

Note that the variables in both chains are in the ascending order (a, b and c, d). According to Algorithm 1 we get:

$$\mathcal{X} = c, b, 1, 0, d, b, 1, 0, 0; \qquad (X = c \cdot b + d \cdot b),$$

where the order is changed (c, d, b).
If the following substitutions (Example 3) are performed for the obtained Binary Chain, the possibility of variable looping will appear.

Example 3

$\alpha = b$
$X = c, b, 1, 0, d, b, 1, 0, 0;$
$\mathcal{A} = d, 0, 1; \quad (b = \overline{d}).$

According to Algorithm 1:

$$\mathcal{X} = c, d, 0, 1, d, d, 0, 1, 0; \qquad \text{(variable } d \text{ loops)}.$$

The correct results of the previous Examples should be:
– for Example 2:

$$\mathcal{X} = b, c, 1, d, 1, 0, 0; \qquad (X = b \cdot c + \bar{b} \cdot d).$$

– for Example 3:

$$\mathcal{X} = c, d, 0, 1, 0 \qquad (X = c \cdot \bar{d}).$$

It appears from the foregoing examples that Algorithm 1 is unfit for use in automatic test generation.

4. Algorithm 2

```
procedure Algorithm2(X, α, A, 𝒳);
begin
 addVariablesOf(X, S);
 addVariablesOf(A, S);
   {S is in ascending order}
 ss:=sizeOf(S);
 for i:=1 to ss do
   for j:=1 to 2^(i-1) do
     𝒳[varPos(i,j,ss)]:=S[i];
   for i:=1 to 2^ss do
     𝒳[constPos(i,ss)]:=value(i,ss,X,A);
end;
```

Such an algorithm makes it possible to obtain the Binary Chain $\mathcal{X}$ fulfilling all the restrictions assumed but it has to be reduced. Yet, it is very time–consuming because of the complicated calculations of functions varPos, constPos and value. Its time complexity is $O(|X|^2 \star |\mathcal{A}|)$.

For example, the first of them is calculated according to the formula:

$$\text{varPos}(i, j, ss) = i + \sum_{n=0}^{n=i-2} \text{bit}At(j, n) \cdot (2^{ss-i+n+2} - 1),$$

where bit$At(j, n)$ means the binary value of n–th bit of the number j.

5. Algorithm 3

We can now present an algorithm which avoids the shortcomings of Algorithms 1 and 2.

```
    procedure Algorithm3(X,α,A,X);
    begin
(1)   addVariablesOf(X,S);
      if α in S then
        if A=const then
(2)       X:=restriction(X,α,const)
        else
          begin
(3)         removeVariable(S,α);
            addVariablesOf(A,S);
              {S is in ascending order}
            ξ:=S[1];
            A1:=restriction(A,v,1);
            A0:=restriction(A,v,0);
(4)         Algorithm3(X,α,A1,X1);
            Algorithm3(X,α,A0,X0);
            binaryChain(ξ,X1,X2,X);
          end
      else
        X:=X;
    end;
```

Step (4) of this algorithm contains the recurrent call to itself. Step (2) contains the conditions for the end of recurrence. The time complexity of it is:

$$O((|X|^{1/2} + |\mathcal{A}|^{1/2}) \star \log(|X|^{3/2}))$$

The calculation of the time complexity of Algorithms 2 and 3 is quite lengthy and was therefore not included in the scope of this paper but intuitively, Algorithm 3 is better.

6. An Example of functioning of Algorithm 3

Let us consider the data from Example 2:

$$(\alpha = a;\quad X = a, b, 1, 0, 0;\quad \mathcal{A} = c, 1, d, 1, 0).$$

1–st call of the Algorithm 3:
1) $S = \{a, b\}$;
2) – ;
3) $S = \{b, c, d\}$. $\xi = b$;
4) $\alpha = a.X = X|_{\xi=1} = a, 1, 0.\mathcal{A} = \mathcal{A}|_{\xi=1} = c, 1, d, 1, 0$;

2–nd call of the Algorithm 3:
1) $S = \{a\}$;
2) – ;
3) $S = \{c, d\}.\xi = c$;
4) $\alpha = a.X = X|_{\xi=1} = a, 1, 0.\mathcal{A} = \mathcal{A}_{\xi=1} = 1$;

3–rd call of the Algorithm 3:
1) $S = \{a\}$;
2) $\mathcal{X} = X|_{\alpha=1} = 1$. Return to call number 2;

2–nd call of the Algorithm 3:
4) $X1 = 1.\alpha = a.X = X|_{\xi=0} = a, 1, 0.\mathcal{A} = \mathcal{A}_{\xi=0} = d, 1, 0$;

4–th call of the Algorithm 3:
1) $S = \{a\}$;
2) – ;
3) $S = \{d\}.\xi = d$;
4) $\alpha = a.X = X|_{\xi=1} = a, 1, 0.\mathcal{A} = \mathcal{A}_{\xi=1} = 1$;

5–th call of the Algorithm 3:
1) $S = \{a\}$;
2) $\mathcal{X} = X|_{\alpha=1} = 1$. Return to call number 4;

4–th call of the Algorithm 3:
4) $X1 = 1.\alpha = a.X = X|_{\xi=0} = a, 1, 0.\mathcal{A} = \mathcal{A}|_{\xi=0} = 0$;

6–th call of the Algorithm 3:
1) $S = \{a\}$;
2) $\mathcal{X} = X|_{\alpha=0} = 0$. Return to call number 4;

4–th call of the Algorithm 3:
4) $X0 = 0$;
5) $\mathcal{X} = d, 1, 0$. Return to call number 2;

2–nd call of the Algorithm 3:
4) $X0 = d, 1, 0$;
5) $\mathcal{X} = c, 1, d, 1, 0$. Return to call number 1;

1–st call of the Algorithm 3:
4) $X1 = c, 1, d, 1, 0.\alpha = a.X = X|_{\xi=0} = 0.\mathcal{A} = \mathcal{A}|_{\xi=0} = c, 1, d, 1, 0$;

7–th call of the Algorithm 3:
1) $S = \{\}.\mathcal{X} = 0$. Return to call number 1;

1–st call of the Algorithm 3:
4) $X0 = 0$;
5) $\mathcal{X} = b, c, 1, d, 1, 0, 0$.

7. Conclusions

The qualities of the Algorithm 3 in comparison with both previous ones are:

a) – Utilization of the calculating chain restriction procedure. This procedure is indispensable in many operations during the test generation and thanks to that more economical use the memory resources is ensured;

b) – The resulting Binary Chain is already reduced, so that the algorithm is much less time–consuming than Algorithm 2:

c) – Complicated numeric calculations are avoided, and the implementing program is quite legible;

d) – Use of recurrence cuts down the length of the implementing program [8].

References

[1] AKERS S.B. Binary decision diagrams.*Proc. IEEE TC* vol.**C–27,**June 1978.

[2] BRYANT R. Graph–Based Algorithms for Boolean Function Manipulation.*Proc. IEEE TC* vol. **C–35,**August 1986.

[3] CZICHON T. Wyznaczanie testów dla układów cyfrowych opisywanych za pomocą proceduralnych języków do opisu sprzętu. Politechnika Warszawska, Ph.D. dissertation, (in Polish) Warszawa 1986.

[4] KINOSITA K., ASADA K., KARACU O. **Logiczeskoje projektirowanije SBIS.**(in Russian) *Mir*, Moskwa 1988.

[5] LIPSETT R., MARSCHNER E., SHAHDAD M. **VHDL – the language.** *IEEE Design & Test*, April 1986.

[6] SAPIECHA K., CZICHON T. Test generation for circuits described in the procedural Hardware Description Languages (HDL's). Int. Conf. Euromicro 86, Venice, pp. 371–379.

[7] SAPIECHA K. **Testowanie i diagnostyka systemów cyfrowych.** (in Polish) *PWN*, Warszawa 1987.

[8] WIRTH N. **Algorytmy + struktury danych = programy.** *WNT*, (in Polish) Warszawa 1980.

Algorytm podstawienia dla łańcuchów binarnych

Artykuł przedstawia pewien algorytm z dziedziny algebry łańcuchów binarnych, używany do opisu funkcjonalnego obwodów cyfrowych w automatycznym generowaniu testów. Pokazano przykład działania tego algorytmu oraz porównanie z innymi algorytmami.

Алгоритм подстановки для бинарных цепей

В статье представлен некоторый алгоритм из области алгебры бинарных цепей, используемый для функционального описания цифровых контуров автоматического генерирования тестов. Представлен пример действия этого алгоритма а также сравнение с другими алгоритмами.