1 Analysis by S-function

In this section, we analyze the BCT of an addition by the S-function technique from [MVDP11]. We also compare this analysis with ours in the submission to explain the differences explicitly.

1.1 Defining the BCT Entry

The original definition of a BCT entry is

$$\begin{aligned}\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) = \#\Big\{ (L, R) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \;\Big|\; & \big((L \boxplus_n R) \oplus \nabla_l \boxminus_n (R \oplus \nabla_r)\big) \\ & \oplus \Big(\big((L \oplus \Delta_l) \boxplus_n (R \oplus \Delta_r)\big) \oplus \nabla_l \boxminus_n (R \oplus \Delta_r \oplus \nabla_r)\Big) = \Delta_l \Big\}.\end{aligned} \tag{1}$$

To use the S-function technique, this formula should be rewritten. Given n-bit words x_1, y_1, Δx, Δy, ∇x, and ∇y, we calculate Δz using

$$x_2 \leftarrow x_1 \oplus \Delta x, \tag{2}$$
$$y_2 \leftarrow y_1 \oplus \Delta y, \tag{3}$$
$$t_1 \leftarrow x_1 + y_1, \tag{4}$$
$$t_2 \leftarrow x_2 + y_2, \tag{5}$$
$$u_1 \leftarrow t_1 \oplus \nabla x, \tag{6}$$
$$v_1 \leftarrow y_1 \oplus \nabla y, \tag{7}$$
$$u_2 \leftarrow t_2 \oplus \nabla x, \tag{8}$$
$$v_2 \leftarrow y_2 \oplus \nabla y, \tag{9}$$
$$z_1 \leftarrow u_1 - v_1, \tag{10}$$
$$z_2 \leftarrow u_2 - v_2, \tag{11}$$
$$\Delta z \leftarrow z_1 \oplus z_2. \tag{12}$$

Then, the BCT entry is redefined as

$$\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) = \# \left\{ (x_1, y_1) \in \mathbb{F}_2^n \times \mathbb{F}_2^n \mid \Delta x = \Delta_l, \Delta y = \Delta_r, \nabla x = \nabla_l, \nabla y = \nabla_r, \Delta z = \Delta_l \right\}.$$

1.2 Constructing the S-Function

Equations (2)-(12) are rewritten on a bit level:

$$x_2[i] \leftarrow x_1[i] \oplus \Delta x[i], \tag{13}$$
$$y_2[i] \leftarrow y_1[i] \oplus \Delta y[i], \tag{14}$$
$$t_1[i] \leftarrow x_1[i] \oplus y_1[i] \oplus c_1[i], \tag{15}$$
$$t_2[i] \leftarrow x_2[i] \oplus y_2[i] \oplus c_2[i], \tag{16}$$
$$c_1[i+1] \leftarrow (x_1[i] + y_1[i] + c_1[i]) \gg 1, \tag{17}$$
$$c_2[i+1] \leftarrow (x_2[i] + y_2[i] + c_2[i]) \gg 1, \tag{18}$$
$$u_1[i] \leftarrow t_1[i] \oplus \nabla x[i], \tag{19}$$
$$v_1[i] \leftarrow y_1[i] \oplus \nabla y[i], \tag{20}$$
$$u_2[i] \leftarrow t_2[i] \oplus \nabla x[i], \tag{21}$$
$$v_2[i] \leftarrow y_2[i] \oplus \nabla y[i], \tag{22}$$
$$z_1[i] \leftarrow u_1[i] \oplus v_1[i] \oplus b_1[i], \tag{23}$$
$$z_2[i] \leftarrow u_2[i] \oplus v_2[i] \oplus b_2[i], \tag{24}$$
$$b_1[i+1] \leftarrow \big((2 + u_1[i] - v_1[i] - b_1[i]) \gg 1\big) \oplus 1, \tag{25}$$
$$b_2[i+1] \leftarrow \big((2 + u_2[i] - v_2[i] - b_2[i]) \gg 1\big) \oplus 1, \tag{26}$$
$$\Delta z[i] \leftarrow z_1[i] \oplus z_2[i], \tag{27}$$

where carries $c_1[0] = c_2[0] = 0$ and borrows $b_1[0] = b_2[0] = 0$. Let the states be

$$\begin{aligned} S[i] &\leftarrow (c_1[i], b_1[i], c_2[i], b_2[i]), \\ S[i+1] &\leftarrow (c_1[i+1], b_1[i+1], c_2[i+1], b_2[i+1]). \end{aligned}$$

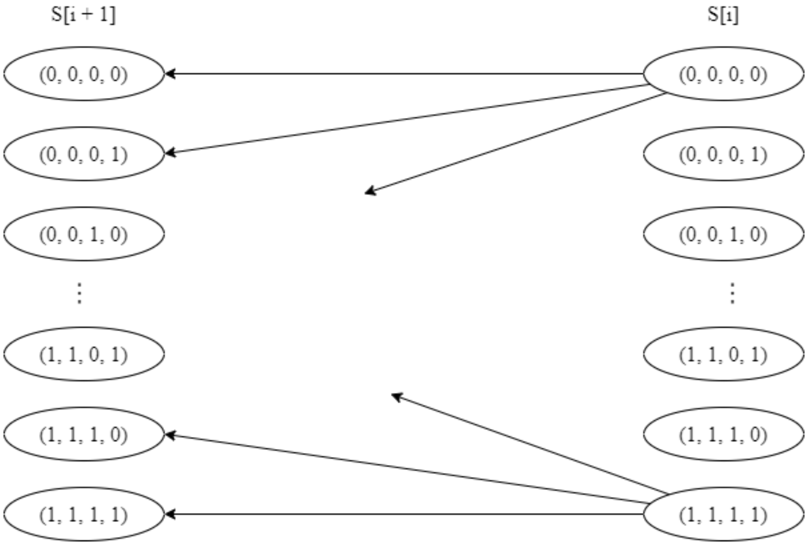


Figure 1: A bipartite graph from the S-function.

Then, Equations (13)-(27) corresponds to the S-function for the BCT of an addition, as the following.

$$(\Delta z[i], S[i+1]) = f(x_1[i], y_1[i], \Delta x[i], \Delta y[i], \nabla x[i], \nabla y[i], S[i]), 0 \le i < n \tag{28}$$

1.3 Computing the Entry

In order to derive the formula for computing the BCT entry, a graph is generated from the S-function (Equation (28)). For $0 \le i \le n$, every state $S[i]$ is represented as a vertex in the graph. Each edge is drawn according to Equation (28). Specifically, for every pair of states, $S[i]$ and $S[i+1]$, if there are values of $x_1[i]$, $y_1[i]$, $\Delta x[i]$, $\Delta y[i]$, $\nabla x[i]$, and $\nabla y[i]$ such that $\Delta z[i]$ is equal to $\Delta x[i]$, then an edge is drawn between vertices $S[i]$ and $S[i+1]$. The resulting graph consists of bipartite graphs each of which contains the vertices $S[i]$ and $S[i+1]$. Figure 1 shows an example of a bipartite graph. Note that, for any bit position i, there are 16 values for $S[i]$, which generates 16 different vertices.

According to [MVDP11], the BCT entry is equal to the number of the paths from $(c_1[0], b_1[0], c_2[0], b_2[0]) = (0, 0, 0, 0)$ to any of the 16 vertices $(c_1[n], b_1[n], c_2[n], b_2[n])$. Let $w[i]$ be $\Delta_l[i] \| \Delta_r[i] \| \nabla_l[i] \| \nabla_r[i]$. Thanks to the nice properties of a bipartite graph, we can use a matrix $A_{w[i]} = [x_{kj}]$ to describe it, where x_{kj} is the number of edges that connect vertices $j = S[i]$ and $k = S[i+1]$. Define a 1×16 matrix $L = [1\ 1\ \ldots\ 1]$ and a 16×1 matrix $C = [1\ 0\ \ldots\ 0]^T$. Then, the formula of the BCT entry is

$$\mathrm{BCT}_n(\Delta_l, \Delta_r, \nabla_l, \nabla_r) = L A_{w[n-1]} A_{w[n-2]} \cdots A_{w[0]} C. \tag{29}$$

Finally, the FSM reduction algorithm in [MVDP11] helps to reduce the number of states.

1.4 Discussion

In this subsection, we explain the differences between our technique and the technique in [MVDP11] as well as the arxtools from [Leu12].

First, our technique focuses on the recursive form of Equation 1 instead of the S-function and the bipartite graphs, which is more elegant. As shown in the previous subsections, the previous technique requires writing down many equations and rewriting them. In Section 1.3, we also have to rely on the bipartite graphs to derive the matrices and the final formula. However, our technique starts by transforming Equation (1) and turns it into a recursive form. This transformation does not require any complicated functions and graphs. Essentially, our idea is from the perspective of dynamic programming. After obtaining the recursive form, the algorithm comes out naturally.

Second, to reduce the number of states, our technique digs out the mathematical property behind the recursive form. The previous technique uses an FSM reduction algorithm to achieve this goal. Indeed, this is a nice idea and can keep the number of states to be the fewest. On the other hand, this algorithm does not tell us what mathematical properties are being used while ours does. Moreover, our computation allows us to find out more properties explained in Section 3.3 of our paper, which lead to the SAT models in Section 4.

Furthermore, the aim of arxtools and that of our technique are completely different. To the best of our knowledge, the arxtools is used to compute the probability of a differential (or boomerang) characteristic and detect incompatibility. It cannot find out a boomerang characteristic with a high probability. Nonetheless, our technique aims to search for such a characteristic. Except the arxtools, we have not found out any other automatic search technique targeting this problem. Therefore, we think that our technique is the first attempt.

In conclusion, our technique is different from the previous ones. It is just similar to the existing techniques, but not identical. In fact, in this scenario, ours has advantages over the others.

[MVDP11] Nicky Mouha, Vesselin Velichkov, Christophe De Cannière, and Bart Preneel. The differential analysis of S-functions. In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, SAC 2010, volume 6544 of LNCS, pages 36–56. Springer, Heidelberg, August 2011.
[Leu12] Analysis of differential attacks in ARX constructions. Gaëtan Leurent, Asiacrypt 2012.

2 Results from ARXTools

In this section, we present the results from the arxtools. The first graph is generated from the command:

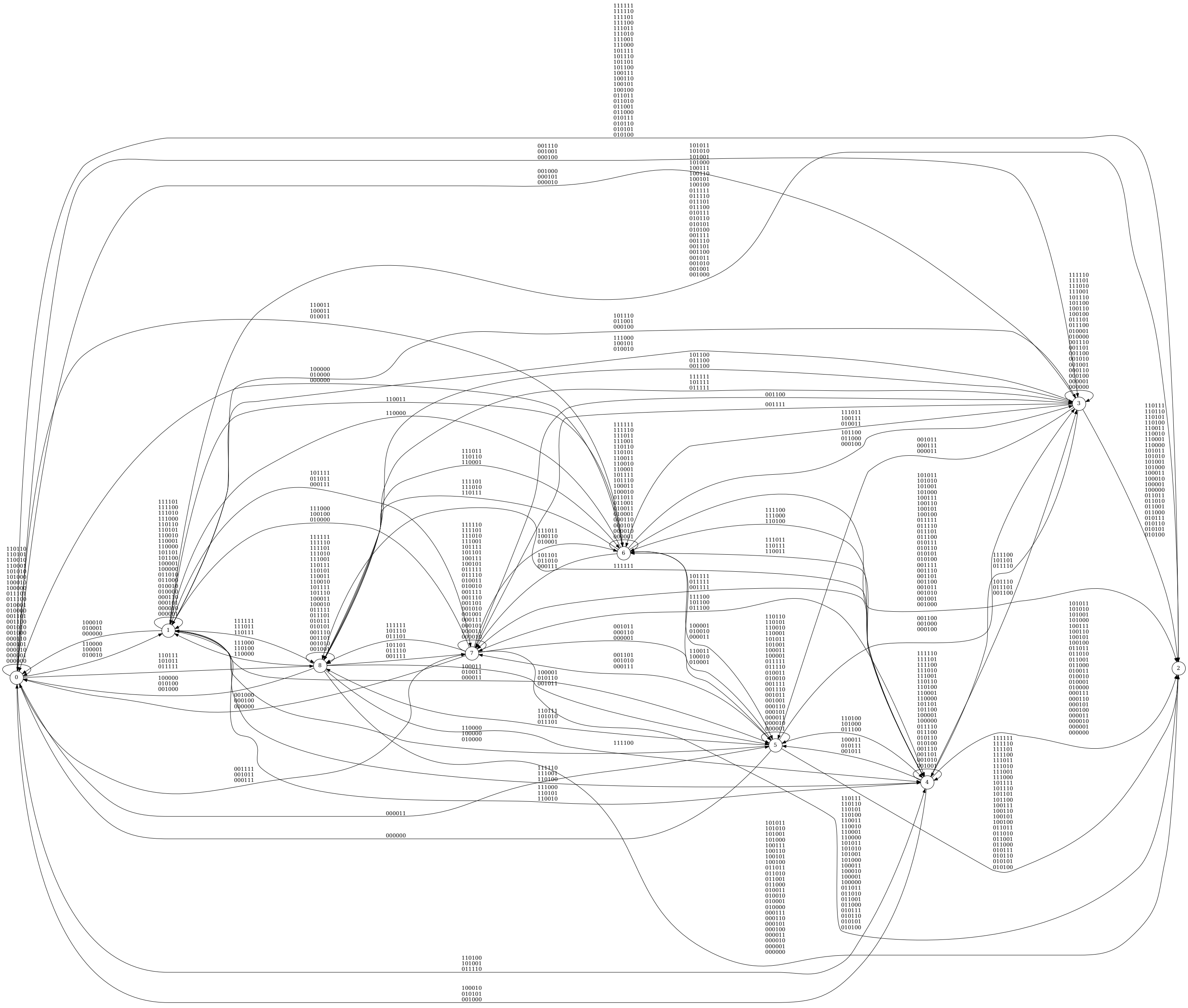
```
build_fsm -e '(((V0+V1)^P2)-(V1^P3))^((((V0^P0)+(V1^P1))^P2)-(V1^P1^P3))==P0' -t -g.
```

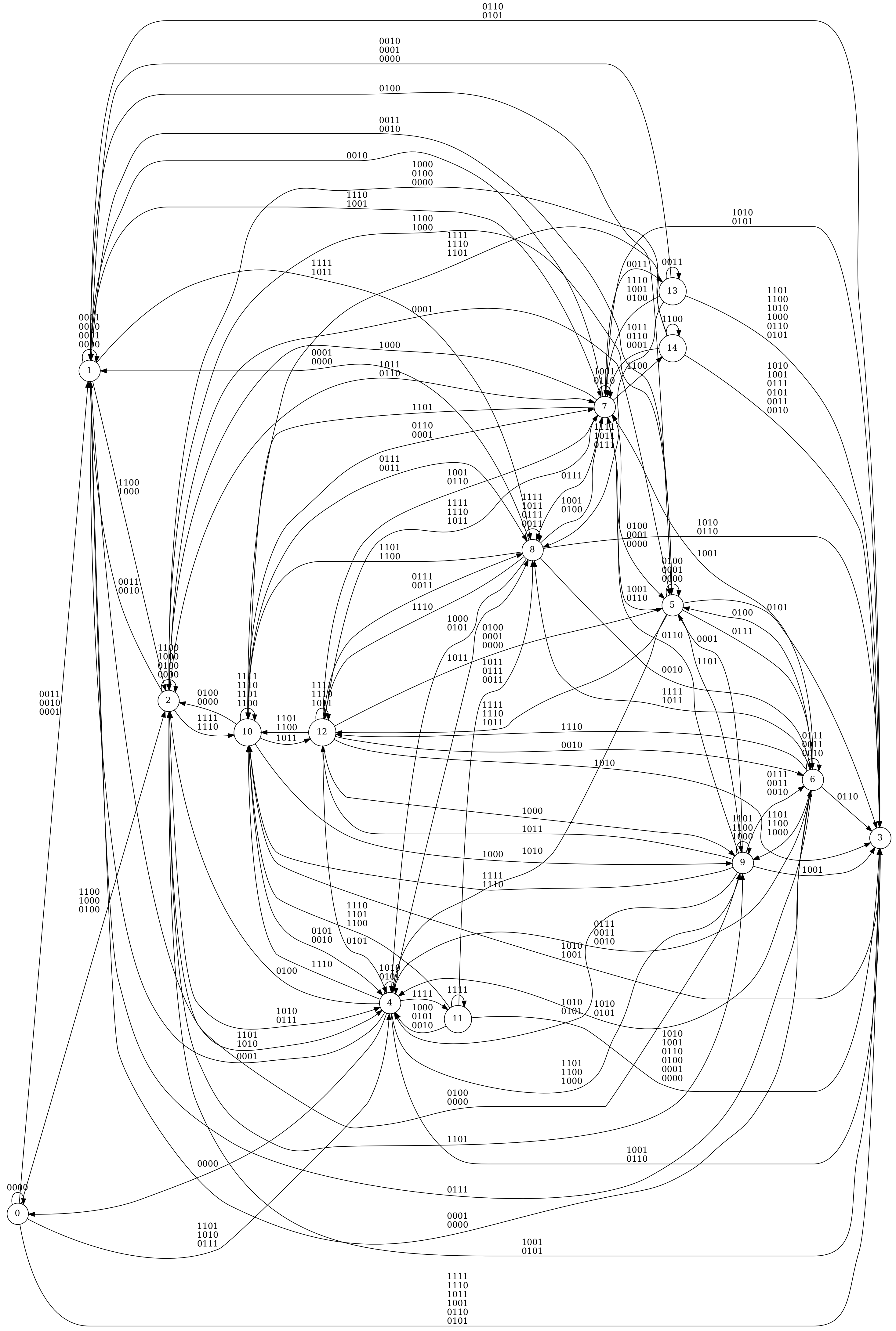
The second graph is generated from the command:

```
build_fsm -e '(((V0+V1)^P2)-(V1^P3))^((((V0^P0)+(V1^P1))^P2)-(V1^P1^P3))==P0' -d -g.
```

We have tried them, but we currently do not find out any possibility to simplify our technique based on these results. It seems that our current computation is easier to derive the SAT models. Thanks for the ideas from Review-#6D.

(Please see the next two pages for the results.)


0
1
2
3
4
5
6
7
8


0110
0101
0010
0001
0000
0100
0011
0010
0010
1000
0100
0000
1110
1001
1010
0101
1100
1000
1111
1110
1101
1111
1011
0011
0011
13
1110
1001
0100
1101
1100
1010
1000
0110
0101
0001
1011
0110
0001
1100
14
0011
0010
0001
0000
1
1000
0001
0000
1011
0110
1001
0110
1100
1010
1001
0111
0101
0011
0010
7
1101
0110
0001
1111
1011
0111
0111
0011
1001
0110
0111
1111
1110
1011
1111
1011
0111
0011
1001
0100
1100
1000
0100
0001
0000
1010
0110
8
1101
1100
1001
0111
0011
0100
0001
0000
1001
0110
1110
5
0101
0011
0010
1000
0101
0100
0100
0001
0000
0111
1100
1000
0100
0000
1011
0110
0001
1011
0111
0011
0010
1101
2
0100
0000
1111
1110
1101
1100
1111
1110
1011
1111
1011
0011
0010
0001
1111
1110
10
1101
1100
1011
12
1110
0010
1010
0111
0011
0010
6
0111
0011
0010
0110
1101
1100
1000
1101
1100
1000
1000
1011
3
1000
9
1010
1001
1111
1110
1100
1000
0100
1110
1101
1100
0111
0011
0010
0101
0010
0101
1010
1001
1110
0100
1010
0101
1111
1010
0101
1010
0101
1010
0111
4
1111
1000
0101
0010
11
1101
1010
0001
1010
1001
0110
0100
0001
0000
1101
1100
1000
0100
0000
1101
0000
1001
0110
0000
0
0111
0001
0000
1101
1010
0111
1001
0101
1111
1110
1011
1001
0110
0101