Satoku Matrix

Release 0.0

Wolfgang Scherer

Apr 18, 2025

Quickstart

See http://sw-amt.ws/README-Satoku.html for an explanation of the satoku matrix.

To get started load `satoku.el`:

Press `C-x C-e` at end of the following expression:

```
(load-file "satoku.el")
```

or press:

```
M-x load-file RET satoku.el RET
```

Then execute the command satoku-insert-matrix:

```
M-x satoku-insert-matrix RET
```

Which will give you a Satoku matrix with a single clause sub-matrix:

```
// |    ||   S||  0|   0|| 1 1 1 |
// +----++----++---+----++-------+
// +-----+----++---+----++-------+
// |     |   0||   |    || 1 * * |
// |     |   1||   |    || * 1 * |
// |     |   2||   |    || * * 1 |
// +-----+----++---+----++-------+
// |:info:| move point inside and press `C-u M-x satoku-mode RET'
```

Position point somewhere inside the matrix and execute:

```
C-u M-x satoku-mode RET
```

This constrains your cursor movements to the matrix and assigns special commands to certain keys. (Note: Outside the matrix, the keys work as usual).

Press `?` for help.

Some commands are prefix commands (e.g. `g`). Press `?` after the prefix command to get help.

To extend the matrix with another 3x3 clause sub-matrix, press `C-u 3 > >`:

```
// |    ||   S||  0|   0|| 1 1 1 | 1 1 1 |
// +----++----++---+----++-------+-------+
// +-----+----++---+----++-------+-------+
// |     |   0||   |    || 1 * * | _ _ _ |
// |     |   1||   |    || * 1 * | _ _ _ |
// |     |   2||   |    || * * 1 | _ _ _ |
// +-----+----++---+----++-------+-------+
// |     |   3||   |    || _ _ _ | 1 * * |
// |     |   4||   |    || _ _ _ | * 1 * |
// |     |   5||   |    || _ _ _ | * * 1 |
// +-----+----++---+----++-------+-------+
// |:info:| move point inside and press `C-u M-x satoku-mode RET'
```

Press `> >` to get a separator line:

```
// |    ||   S||  0|   0|| 1 1 1 | 1 1 1 ||
// +----++----++---+----++-------+-------++
// +-----+----++---+----++-------+-------++
// |     |   0||   |    || 1 * * | _ _ _ ||
// |     |   1||   |    || * 1 * | _ _ _ ||
// |     |   2||   |    || * * 1 | _ _ _ ||
// +-----+----++---+----++-------+-------++
// |     |   3||   |    || _ _ _ | 1 * * ||
// |     |   4||   |    || _ _ _ | * 1 * ||
// |     |   5||   |    || _ _ _ | * * 1 ||
```

(continues on next page)

(continued from previous page)

```
// +-----+----++---+----++-------+-------++
// +-----+----++---+----++-------+-------++
// |:info:| move point inside and press `C-u M-x satoku-mode RET'
```

Press `C-u 2 > >` to get a 2x2 clause sub-matrix, repeat, and repeat again (hint: you can use keyboard macros).

Normal keys are disabled inside the matrix (but not the legend), use `0` to mark selection conflicts, − to clear a cell. (Mirror cells are automatically changed accordingly.)

Do not use `1` to mark required selections. Exclude all other selections with `0` instead:

```
// |    ||   S||  0|   0|| 1 1 1 | 1 1 1 || 1 1 | 1 1 | 1 1 |
// +----++----++---+----++-------+-------++-----+-----+-----+
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   0||   |    || 1 * * | 0 _ _ || _ 0 | _ _ | _ _ |
// |     |   1||   |    || * 1 * | _ _ 0 || _ _ | _ _ | _ _ |
// |     |   2||   |    || * * 1 | _ _ 0 || _ _ | _ _ | _ _ |
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   3||   |    || 0 _ _ | 1 * * || 0 _ | _ _ | _ _ |
// |     |   4||   |    || _ _ _ | * 1 * || _ _ | _ _ | _ _ |
// |     |   5||   |    || _ 0 0 | * * 1 || _ _ | _ _ | _ _ |
// +-----+----++---+----++-------+-------++-----+-----+-----+
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   6||   |    || _ _ _ | 0 _ _ || 1 * | _ _ | _ _ |  a
// |     |   7||   |    || 0 _ _ | _ _ _ || * 1 | _ _ | _ _ | ¬a
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   8||   |    || _ _ _ | _ _ _ || _ _ | 1 * | _ _ |  b
// |     |   9||   |    || _ _ _ | _ _ _ || _ _ | * 1 | _ _ | ¬b
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |  10||   |    || _ _ _ | _ _ _ || _ _ | _ _ | 1 * |  c
// |     |  11||   |    || _ _ _ | _ _ _ || _ _ | _ _ | * 1 | ¬c
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |:info:| move point inside and press `C-u M-x satoku-mode RET'
```

Then press `r u` to run the requirements update algorithm, which detects and fills in the *hard one* requirements.

If you did use `1` and are no longer sure, whether the matrix conditions are proper, press `r c` to clear all *hard one* requirements. Then press `r u` to run the requirements update algorithm.

```
// |    ||   S||  0|   0|| 1 1 1 | 1 1 1 || 1 1 | 1 1 | 1 1 |
// +----++----++---+----++-------+-------++-----+-----+-----+
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   0||   |    || 1 * * | 0 _ _ || 1 0 | _ _ | _ _ |
// |     |   1||   |    || * 1 * | _ _ 0 || _ _ | _ _ | _ _ |
// |     |   2||   |    || * * 1 | _ _ 0 || _ _ | _ _ | _ _ |
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   3||   |    || 0 _ _ | 1 * * || 0 1 | _ _ | _ _ |
// |     |   4||   |    || _ _ _ | * 1 * || _ _ | _ _ | _ _ |
// |     |   5||   |    || 1 0 0 | * * 1 || 1 0 | _ _ | _ _ |
// +-----+----++---+----++-------+-------++-----+-----+-----+
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   6||   |    || _ _ _ | 0 _ _ || 1 * | _ _ | _ _ |  a
// |     |   7||   |    || 0 _ _ | _ _ 0 || * 1 | _ _ | _ _ | ¬a
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |   8||   |    || _ _ _ | _ _ _ || _ _ | 1 * | _ _ |  b
// |     |   9||   |    || _ _ _ | _ _ _ || _ _ | * 1 | _ _ | ¬b
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |     |  10||   |    || _ _ _ | _ _ _ || _ _ | _ _ | 1 * |  c
// |     |  11||   |    || _ _ _ | _ _ _ || _ _ | _ _ | * 1 | ¬c
// +-----+----++---+----++-------+-------++-----+-----+-----+
// |:info:| move point inside and press `C-u M-x satoku-mode RET'
```

Abstract

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODE BLOCKS

CHAPTER

ONE

INTRODUCTION

CHAPTER

TWO

BIT COUNTER

A bit counter for 3 bits is presented as an example for determining the Blake normal form (sum of prime implicants) of a logical formula in conjunctive normal form using the *satoku matrix*.

2.1 Truth Table

For 3 input bits A, B and C, the outputs S_0 and S_1 shall reflect the number of 1 bits at the inputs as a binary number, where $S_0 = 2^0$ and $S_1 = 2^1$.

In table 2.1, the definitions for the logical functions of S_0 and S_1 are shown.

table 2.1: Truth table for bit counter

A	B	C	S_1	S_0
0	**0**	**0**	**0**	**0**
0	**0**	**1**	**0**	**1**
0	**1**	**0**	**0**	**1**
0	**1**	**1**	**1**	**0**
1	**0**	**0**	**0**	**1**
1	**0**	**1**	**1**	**0**
1	**1**	**0**	**1**	**0**
1	**1**	**1**	**1**	**1**

2.2 Processing S_0

The conjunctive normal form (*CNF*) for S_0 is derived from truth table table 2.1 by constructing an *implicate*, which is a disjunctive clause that becomes false for each of the inputs, when the output is false:

$$\begin{aligned} S_0 \quad = \quad & (\ A \vee \ B \vee \ C) \wedge \\ & (\ A \vee \neg B \vee \neg C) \wedge \\ & (\neg A \vee \ B \vee \neg C) \wedge \\ & (\neg A \vee \neg B \vee \ C) \end{aligned} \tag{2.1}$$

Complementary Assignment (distributive expansion, multiplication) of (2.1) delivers a special disjunctive normal form (*DNF*) containing all prime *implicant*s called the Blake canonical form (*BCF*) for S_0:

$$\begin{aligned} S_0 \quad = \quad & (\neg A \wedge \neg B \wedge \ C) \vee \\ & (\neg A \wedge \ B \wedge \neg C) \vee \\ & (\ A \wedge \neg B \wedge \neg C) \vee \\ & (\ A \wedge \ B \wedge \ C) \end{aligned} \tag{2.2}$$

Complementary Assignment of *BCF* (2.2) delivers the set of prime *implicate*s for S_0:

$$\begin{aligned} S_0 \quad = \quad & (\neg A \vee \neg B \vee \ C) \\ & (\neg A \vee \ B \vee \neg C) \wedge \\ & (\ A \vee \neg B \vee \neg C) \wedge \\ & (\ A \vee \ B \vee \ C) \wedge \end{aligned} \tag{2.3}$$

Adding a clause for each variable to (2.1) containing the variable and its negation delivers the extended *CNF* for mapping to a *satoku matrix*:

$$
\begin{aligned}
S_0 \quad = \quad & (\ \ A \vee \ \ B \vee \ \ C) \wedge \\
& (\ \ A \vee \neg B \vee \neg C) \wedge \\
& (\neg A \vee \ \ B \vee \neg C) \wedge \\
& (\neg A \vee \neg B \vee \ \ C) \wedge \\
& (\ \ A \vee \neg A) \wedge \\
& (\ \ B \vee \neg B) \wedge \\
& (\ \ C \vee \neg C)
\end{aligned}
\tag{2.4}
$$

A *satoku matrix* $\mathbb{S}$ is based on an inverted symmetric *adjacency matrix*.

In figure 2.1 the *satoku matrix* $\mathbb{S}_0$ is mapped from the extended *CNF* formula (2.4) for S_0, analogous to mapping a SAT problem to an independent set problem [MOUNT2012], pp. 92.

The clauses are mapped to $r_{0_{0_0}} \dots r_{3_{2_3}}$, the variables are mapped to $r_{4_{0_4}} \dots r_{6_{1_6}}$, The region at $r_{0_{0_4}} \dots r_{3_{2_6}}$ reflects the originally required variables for each input *CNF*.

P	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	0 − −	1 0	− −	− −	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− 0 −	− −	1 0	− −	$B+$
s_{0_2}	∘ ∘ 1	− − 0	− − 0	− − −	− −	− −	1 0	C
s_{1_0}	− − −	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	$A+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	$B'+$
s_{1_2}	− − 0	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	C'
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	$B+$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	$A'+$
s_{3_1}	− 0 −	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	$B'+$
s_{3_2}	− − −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	C
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	A'
s_{5_0}	− − −	− 0 −	− − −	− 0 −	− −	1 ∘	− −	$B+$
s_{5_1}	− 0 −	− − −	− 0 −	− − −	− −	∘ 1	− −	B'
s_{6_0}	− − −	− − 0	− − 0	− − −	− −	− −	1 ∘	$C+$
s_{6_1}	− − 0	− − −	− − −	− − 0	− −	− −	∘ 1	C'

figure 2.1: *Satoku matrix** $\mathbb{S}_0$ for S_0

Since *state rows* s_{0_0} and s_{1_0} are *independent*, because *conflict relationship* $s_{0_{0_{1_0}}} \neq 0$ and *conflict relationship* $s_{1_{0_{0_0}}} \neq 0$ and the relevant *conflict subsequences* $r_{0_{0_2}} \dots r_{0_{0_3}}$ and $r_{1_{0_2}} \dots r_{1_{0_3}}$ are equal, they can be *joined* by making each other *required*, i.e setting *conflict relationships* $s_{0_{0_{1_1}}} := 0, s_{0_{0_{1_2}}} := 0$, which makes *conflict relationship* $s_{0_{0_{1_0}}}$ *required*, and setting *conflict relationships* $s_{1_{0_{0_1}}} := 0, s_{1_{0_{0_2}}} := 0$, which makes *conflict relationship* $s_{1_{0_{0_0}}}$ *required*. (See figure 2.2).

P	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	− 0 0	0 − −	0 − −	1 0	− −	− −	$A+$
s_{0_1}	∘ 1 ∘	0 0 −	− − −	− 0 −	− −	1 0	− −	$B+$
s_{0_2}	∘ ∘ 1	0 − 0	− − 0	− − −	− −	− −	1 0	C
s_{1_0}	− 0 0	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	$A+$
s_{1_1}	0 0 −	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	$B'+$
s_{1_2}	0 − 0	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	C'
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	$B+$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	$A'+$
s_{3_1}	− 0 −	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	$B'+$
s_{3_2}	− − −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	C
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	A'
s_{5_0}	− − −	− 0 −	− − −	− 0 −	− −	1 ∘	− −	$B+$
s_{5_1}	− 0 −	− − −	− 0 −	− − −	− −	∘ 1	− −	B'
s_{6_0}	− − −	− − 0	− − 0	− − −	− −	− −	1 ∘	$C+$
s_{6_1}	− − 0	− − −	− − −	− − 0	− −	− −	∘ 1	C'

figure 2.2: *State rows* s_{0_0} and s_{1_0} *joined* in $\mathbb{S}_0$

After *consolidating* the *satoku matrix* $\mathbb{S}_0$ by propagating the *conflict relationships* of *required* states, it presents *state rows* s_{0_0} and s_{1_0} *joined* as shown in figure 2.3.

Additional consequences from the advance decision were detected during *consolidation* in *state rows* $s_{0_1}, s_{0_2}, s_{1_1}, s_{1_2}$.

This technique is an arbitrary (but not random!) advance decision to select a state when another state is selected and vice versa. The decision has no consequence on the satisfiability of the problem, since both states are equal at the time of the advance decision. Note, that these advance decisions can only be made in a *consolidated satoku matrix* $\mathbb{S}$. Therefore, it is not valid to *join* more than 2 states at a time.

Also note, that assignments to variables have no strong connection to selections from a clause. I.e., just because one literal of a clause becomes true, it does not preclude any other literal of the clause becoming true. Further, the advance decision does not affect the current global state of any variables.

P	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	1 0 0	0 − −	0 − −	1 0	− −	− −	$A+$
s_{0_1}	∘ 1 ∘	0 0 1	− − −	1 0 0	0 1	1 0	0 1	$A'BC'+$
s_{0_2}	∘ ∘ 1	0 1 0	1 0 0	− − −	0 1	0 1	1 0	$A'B'C$
s_{1_0}	1 0 0	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	$A+$
s_{1_1}	0 0 1	∘ 1 ∘	1 0 0	− − −	0 1	0 1	1 0	$A'B'C+$
s_{1_2}	0 1 0	∘ ∘ 1	− − −	1 0 0	0 1	1 0	0 1	$A'BC'$
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	$A'+$
s_{2_1}	− − 0	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	$B+$
s_{2_2}	− − 0	− 0 −	∘ ∘ 1	− − 0	− −	− −	0 1	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	$A'+$
s_{3_1}	− 0 −	− − 0	− 0 −	∘ 1 ∘	− −	0 1	− −	$B'+$
s_{3_2}	− 0 −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	C
s_{4_0}	1 0 0	1 0 0	0 − −	0 − −	1 ∘	− −	− −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	A'
s_{5_0}	− − 0	− 0 −	− − −	− 0 −	− −	1 ∘	− −	$B+$
s_{5_1}	− 0 −	− − 0	− 0 −	− − −	− −	∘ 1	− −	B'
s_{6_0}	− 0 −	− − 0	− − 0	− − −	− −	− −	1 ∘	$C+$
s_{6_1}	− − 0	− 0 −	− − −	− − 0	− −	− −	∘ 1	C'

figure 2.3: *State rows* s_{0_0} and s_{1_0} *joined* and *consolidated* in $\mathbb{S}_0$

Since *state rows* s_{2_0} and s_{3_0} are *independent* and their relevant *conflict subsequences* are equal, they can be *joined* by making each other required as shown in figure 2.4.

P	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	1 0 0	0 − −	0 − −	1 0	− −	− −	$A+$
s_{0_1}	∘ 1 ∘	0 0 1	− − −	1 0 0	0 1	1 0	0 1	$A'BC'+$
s_{0_2}	∘ ∘ 1	0 1 0	1 0 0	− − −	0 1	0 1	1 0	$A'B'C$
s_{1_0}	1 0 0	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	$A+$
s_{1_1}	0 0 1	∘ 1 ∘	1 0 0	− − −	0 1	0 1	1 0	$A'B'C+$
s_{1_2}	0 1 0	∘ ∘ 1	− − −	1 0 0	0 1	1 0	0 1	$A'BC'$
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− 0 0	0 1	− −	− −	$A'+$
s_{2_1}	− − 0	− 0 −	∘ 1 ∘	0 0 −	− −	1 0	− −	$B+$
s_{2_2}	− − 0	− 0 −	∘ ∘ 1	0 − 0	− −	− −	0 1	C'
s_{3_0}	0 − −	0 − −	− 0 0	1 ∘ ∘	0 1	− −	− −	$A'+$
s_{3_1}	− 0 −	− − 0	0 0 −	∘ 1 ∘	− −	0 1	− −	$B'+$
s_{3_2}	− 0 −	− − 0	0 − 0	∘ ∘ 1	− −	− −	1 0	C
s_{4_0}	1 0 0	1 0 0	0 − −	0 − −	1 ∘	− −	− −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	A'
s_{5_0}	− − 0	− 0 −	− − −	− 0 −	− −	1 ∘	− −	$B+$
s_{5_1}	− 0 −	− − 0	− 0 −	− − −	− −	∘ 1	− −	B'
s_{6_0}	− 0 −	− − 0	− − 0	− − −	− −	− −	1 ∘	$C+$
s_{6_1}	− − 0	− 0 −	− − −	− − 0	− −	− −	∘ 1	C'

figure 2.4: *State rows* s_{2_0} and s_{3_0} *joined* in $\mathbb{S}_0$

Conflict propagation during *consolidation* has affected the entire *satoku matrix* $\mathbb{S}_0$ as shown in figure 2.5. *Cell* c_{0_1} shows that all *state rows* from *cell* c_{1_1} are *required* and therefore *cell matrix row* c_1 can be removed from the *satoku matrix*. The same holds for *cell row* c_3 based on the *required conflict relationships* in *cell* c_{2_3}.

P	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	1 0 0	0 − −	0 − −	1 0	− −	− −	$A+$
s_{0_1}	∘ 1 ∘	0 0 1	1 0 0	1 0 0	0 1	1 0	0 1	$A'BC'+$
s_{0_2}	∘ ∘ 1	0 1 0	1 0 0	1 0 0	0 1	0 1	1 0	$A'B'C$
s_{1_0}	1 0 0	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	$A+$
s_{1_1}	0 0 1	∘ 1 ∘	1 0 0	1 0 0	0 1	0 1	1 0	$A'B'C+$
s_{1_2}	0 1 0	∘ ∘ 1	1 0 0	1 0 0	0 1	1 0	0 1	$A'BC'$
s_{2_0}	0 − −	0 − −	1 ∘ ∘	1 0 0	0 1	− −	− −	$A'+$
s_{2_1}	1 0 0	1 0 0	∘ 1 ∘	0 0 1	1 0	1 0	1 0	$ABC+$
s_{2_2}	1 0 0	1 0 0	∘ ∘ 1	0 1 0	1 0	0 1	0 1	$AB'C'$
s_{3_0}	0 − −	0 − −	1 0 0	1 ∘ ∘	0 1	− −	− −	$A'+$
s_{3_1}	1 0 0	1 0 0	0 0 1	∘ 1 ∘	1 0	0 1	0 1	$AB'C'+$
s_{3_2}	1 0 0	1 0 0	0 1 0	∘ ∘ 1	1 0	1 0	1 0	ABC
s_{4_0}	1 0 0	1 0 0	0 − −	0 − −	1 ∘	− −	− −	$A+$
s_{4_1}	0 − −	0 − −	1 0 0	1 0 0	∘ 1	− −	− −	A'
s_{5_0}	− − 0	− 0 −	− − 0	− 0 −	− −	1 ∘	− −	$B+$
s_{5_1}	− 0 −	− − 0	− 0 −	− − 0	− −	∘ 1	− −	B'
s_{6_0}	− 0 −	− − 0	− − 0	− 0 −	− −	− −	1 ∘	$C+$
s_{6_1}	− − 0	− 0 −	− 0 −	− − 0	− −	− −	∘ 1	C'

figure 2.5: *State rows* s_{2_0} and s_{3_0} *joined* and *consolidated* in $\mathbb{S}_0$

In figure 2.6 the redundant *cell matrix rows* are removed.

Counting the *possible conflict relationships* in *cell* c_{0_1} reveals that *merging* both *cell matrix rows* c_0 and c_1 will only result in a maximum of 4 *state rows* in *cell matrix row* c_5.

Cells c_{5_0} and c_{5_1} have been prepared to *require* all combinations of *state rows* allowed by c_{0_1}.

P	− − −	− − −	− −	− −	− −	− − − −	
s_{0_0}	1 ∘ ∘	0 − −	1 0	− −	− −	− − 0 0	$A+$
s_{0_1}	∘ 1 ∘	1 0 0	0 1	1 0	0 1	0 0 − 0	$A'B'C+$
s_{0_2}	∘ ∘ 1	1 0 0	0 1	0 1	1 0	0 0 0 −	$A'BC'$
s_{1_0}	0 − −	1 ∘ ∘	0 1	− −	− −	0 0 − −	$A'+$
s_{1_1}	1 0 0	∘ 1 ∘	1 0	1 0	1 0	− 0 0 0	$ABC+$
s_{1_2}	1 0 0	∘ ∘ 1	1 0	0 1	0 1	0 − 0 0	$AB'C'$
s_{2_0}	1 0 0	0 − −	1 ∘	− −	− −	− − − −	$A+$
s_{2_1}	0 − −	1 0 0	∘ 1	− −	− −	− − − −	A'
s_{3_0}	− − 0	− − 0	− −	1 ∘	− −	− − − −	$B+$
s_{3_1}	− 0 −	− 0 −	− −	∘ 1	− −	− − − −	B'
s_{4_0}	− 0 −	− − 0	− −	− −	1 ∘	− − − −	$C+$
s_{4_1}	− − 0	− 0 −	− −	− −	∘ 1	− − − −	C'
s_{5_0}	− 0 0	0 − 0	− −	− −	− −	1 ∘ ∘ ∘	
s_{5_1}	− 0 0	0 0 −	− −	− −	− −	∘ 1 ∘ ∘	
s_{5_2}	0 − 0	− 0 0	− −	− −	− −	∘ ∘ 1 ∘	
s_{5_3}	0 0 −	− 0 0	− −	− −	− −	∘ ∘ ∘ 1	

figure 2.6: Redundancies removed from $\mathbb{S}_0$

In figure 2.7 the *cell matrix rows* c_0 and c_1 have been *merged* and *satoku matrix* $\mathbb{S}_0$ reveals a set of *implicant*s for S_0 in *cell matrix row* c_5 at range $r_{5_{0_2}} \ldots r_{5_{3_4}}$. It is further possible to remove the *cell matrix rows* c_0 and c_1 from *satoku matrix* $\mathbb{S}_0$ since they are fully *absorbed* by *cells* c_{5_0} and c_{5_1}. This results in a net decrease of the matrix size. Finally, a *state row* permutation has been prepared in *cell* c_{6_5}.

P	− − −	− − −	− −	− −	− −	− − − −		
s_{0_0}	1 ∘ ∘	0 − −	1 0	− −	− −	− − 0 0	− − − −	$A+$
s_{0_1}	∘ 1 ∘	1 0 0	0 1	1 0	0 1	0 0 1 0	− − − −	$A'B'C+$
s_{0_2}	∘ ∘ 1	1 0 0	0 1	0 1	1 0	0 0 0 1	− − − −	$A'BC'$
s_{1_0}	0 − −	1 ∘ ∘	0 1	− −	− −	0 0 − −	− − − −	$A'+$
s_{1_1}	1 0 0	∘ 1 ∘	1 0	1 0	1 0	1 0 0 0	− − − −	$ABC+$
s_{1_2}	1 0 0	∘ ∘ 1	1 0	0 1	0 1	0 1 0 0	− − − −	$AB'C'$
s_{2_0}	1 0 0	0 − −	1 ∘	− −	− −	− − 0 0	− − − −	$A+$
s_{2_1}	0 − −	1 0 0	∘ 1	− −	− −	0 0 − −	− − − −	A'
s_{3_0}	− − 0	− − 0	− −	1 ∘	− −	− 0 − 0	− − − −	$B+$
s_{3_1}	− 0 −	− 0 −	− −	∘ 1	− −	0 − 0 −	− − − −	B'
s_{4_0}	− 0 −	− − 0	− −	− −	1 ∘	− 0 0 −	− − − −	$C+$
s_{4_1}	− − 0	− 0 −	− −	− −	∘ 1	0 − − 0	− − − −	C'
s_{5_0}	1 0 0	0 1 0	1 0	1 0	1 0	1 ∘ ∘ ∘	0 0 0 −	$ABC+$
s_{5_1}	1 0 0	0 0 1	1 0	0 1	0 1	∘ 1 ∘ ∘	0 0 − 0	$AB'C'+$
s_{5_2}	0 1 0	1 0 0	0 1	1 0	0 1	∘ ∘ 1 ∘	0 − 0 0	$A'BC'+$
s_{5_3}	0 0 1	1 0 0	0 1	0 1	1 0	∘ ∘ ∘ 1	− 0 0 0	$A'B'C$
s_{6_0}	− − −	− − −	− −	− −	− −	0 0 0 −	1 ∘ ∘ ∘	
s_{6_1}	− − −	− − −	− −	− −	− −	0 0 − 0	∘ 1 ∘ ∘	
s_{6_2}	− − −	− − −	− −	− −	− −	0 − 0 0	∘ ∘ 1 ∘	
s_{6_3}	− − −	− − −	− −	− −	− −	− 0 0 0	∘ ∘ ∘ 1	

figure 2.7: *Cell matrix rows* c_0 and c_1 *merged* in $\mathbb{S}_0$, implicants at range $r_{5_{0_2}} \ldots r_{5_{3_4}}$

In figure 2.8, *cell matrix rows* c'_0 and c'_1 have been removed from *satoku matrix* $\mathbb{S}_0$. *Consolidation* has generated a *state row* permutation of *cell matrix row* c_3 in *cell matrix row* c_4. The set of conjuctions defined at range $r_{4_{0_0}} \ldots r_{4_{3_2}}$ is now identical to the *BCF* for S_0 in (2.2).

P	− −	− −	− −	− − − −	− − − −	
s_{0_0}	1 ∘	− −	− −	− − 0 0	0 0 − −	$A+$
s_{0_1}	∘ 1	− −	− −	0 0 − −	− − 0 0	A'
s_{1_0}	− −	1 ∘	− −	− 0 − 0	0 − 0 −	$B+$
s_{1_1}	− −	∘ 1	− −	0 − 0 −	− 0 − 0	B'
s_{2_0}	− −	− −	1 ∘	− 0 0 −	− 0 0 −	$C+$
s_{2_1}	− −	− −	∘ 1	0 − − 0	0 − − 0	C'
s_{3_0}	1 0	1 0	1 0	1 ∘ ∘ ∘	0 0 0 1	$ABC+$
s_{3_1}	1 0	0 1	0 1	∘ 1 ∘ ∘	0 0 1 0	$AB'C'+$
s_{3_2}	0 1	1 0	0 1	∘ ∘ 1 ∘	0 1 0 0	$A'BC'+$
s_{3_3}	0 1	0 1	1 0	∘ ∘ ∘ 1	1 0 0 0	$A'B'C$
s_{4_0}	0 1	0 1	1 0	0 0 0 1	1 ∘ ∘ ∘	$A'B'C$
s_{4_1}	0 1	1 0	0 1	0 0 1 0	∘ 1 ∘ ∘	$A'BC'+$
s_{4_2}	1 0	0 1	0 1	0 1 0 0	∘ ∘ 1 ∘	$AB'C'+$
s_{4_3}	1 0	1 0	1 0	1 0 0 0	∘ ∘ ∘ 1	$ABC+$

figure 2.8: *Cell matrix rows* c'_0 and c'_1 removed, c_3 permuted into c_4, *BCF* at range $r_{4_{0_0}} \ldots r_{4_{3_2}}$ in $\mathbb{S}_0$

2.3 Processing S_1

The conjunctive normal form (*CNF*) for S_1 is derived from truth table table 2.1:

$$\begin{aligned} S_1 \quad = \quad & (\ A \vee \ B \vee \ C) \wedge \\ & (\ A \vee \ B \vee \neg C) \wedge \\ & (\ A \vee \neg B \vee \ C) \wedge \\ & (\neg A \vee \ B \vee \ C) \end{aligned} \tag{2.5}$$

Complementary Assignment (distributive expansion, multiplication) of (2.5) delivers a special disjunctive normal form (*DNF*) containing all prime *implicant*s called the Blake canonical form (*BCF*) for S_1:

$$\begin{aligned} S_1 \quad = \quad & (\ B \wedge \ C) \vee \\ & (\ A \wedge \ C) \vee \\ & (\ A \wedge \ B) \end{aligned} \tag{2.6}$$

Complementary Assignment of *BCF* (2.6) delivers the set of prime *implicate*s for S_1:

$$\begin{array}{rcl} S_1 & = & (\ B \vee\ \ C) \wedge \\ & & (\ A \vee\ \ C) \wedge \\ & & (\ A \vee\ \ B) \end{array} \tag{2.7}$$

Adding a clause for each variable to (2.5) containing the variable and its negation delivers the extended *CNF* for mapping to a *satoku matrix*:

$$\begin{array}{l} (\ A \vee\ \ B \vee\ \ C) \wedge \\ \qquad\qquad\qquad\qquad (\ A \vee\ \ B \vee \neg C) \wedge \\ \qquad\qquad\qquad\qquad (\ A \vee \neg B \vee\ \ C) \wedge \\ \qquad\qquad\qquad\qquad (\neg A \vee\ \ B \vee\ \ C) \\ \qquad\qquad\qquad\qquad (\ A \vee \neg A) \wedge \\ \qquad\qquad\qquad\qquad (\ B \vee \neg B) \wedge \\ \qquad\qquad\qquad\qquad (\ C \vee \neg C) \end{array} \tag{2.8}$$

In figure 2.9 the *satoku matrix* $\mathbb{S}_1$ is mapped from the *CNF* formula (2.8) for S_1.

P	− − −	− − −	− − −	− − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	− − −	− − −	0 − −	1 0	− −	− −	$A+$
s_{0_1}	∘ 1 ∘	− − −	− 0 −	− − −	− −	1 0	− −	$B+$
s_{0_2}	∘ ∘ 1	− − 0	− − −	− − −	− −	− −	1 0	C
s_{1_0}	− − −	1 ∘ ∘	− − −	0 − −	1 0	− −	− −	$A+$
s_{1_1}	− − −	∘ 1 ∘	− 0 −	− − −	− −	1 0	− −	$B+$
s_{1_2}	− − 0	∘ ∘ 1	− − 0	− − 0	− −	− −	0 1	C'
s_{2_0}	− − −	− − −	1 ∘ ∘	0 − −	1 0	− −	− −	$A+$
s_{2_1}	− 0 −	− 0 −	∘ 1 ∘	− 0 −	− −	0 1	− −	$B'+$
s_{2_2}	− − −	− − 0	∘ ∘ 1	− − −	− −	− −	1 0	C
s_{3_0}	0 − −	0 − −	0 − −	1 ∘ ∘	0 1	− −	− −	$A'+$
s_{3_1}	− − −	− − −	− 0 −	∘ 1 ∘	− −	1 0	− −	$B+$
s_{3_2}	− − −	− − 0	− − −	∘ ∘ 1	− −	− −	1 0	C
s_{4_0}	− − −	− − −	− − −	0 − −	1 ∘	− −	− −	$A+$
s_{4_1}	0 − −	0 − −	0 − −	− − −	∘ 1	− −	− −	A'
s_{5_0}	− − −	− − −	− 0 −	− − −	− −	1 ∘	− −	$B+$
s_{5_1}	− 0 −	− 0 −	− − −	− 0 −	− −	∘ 1	− −	B'
s_{6_0}	− − −	− − 0	− − −	− − −	− −	− −	1 ∘	$C+$
s_{6_1}	− − 0	− − −	− − 0	− − 0	− −	− −	∘ 1	C'

figure 2.9: *Satoku matrix* $\mathbb{S}_1$ for S_1

In figure 2.10, *Cell row* $r_{0_{0_3}}$ is a *minor conflict cell row*, $\mathrm{MCCR}(r_{i_{j_g}}) \equiv |\mathrm{P}(r_{i_{j_g}})| = 2$, containing 2 *possible conflict relationships*. The consequences can be inspected by *merging state row* s_{0_0} with *state row* s_{3_1}, which is prepared in *cell rows* $r_{7_{0_0}}$ and $r_{7_{0_3}}$, and by *merging state row* s_{0_0} with *state row* s_{3_2}, which is prepared in *cell rows* $r_{7_{1_0}}$ and $r_{7_{1_3}}$. *Cell row* $r_{7_{2_0}}$ expresses the fact that alternatively s_{0_1} or s_{0_2} can become *required*.

P	− − −	− − −	− − −	− − −	− −	− −	− −	− − −	
s_{0_0}	1 ∘ ∘	− − −	− − −	0 − −	1 0	− −	− −	− − 0	$A+$
s_{0_1}	∘ 1 ∘	− − −	− 0 −	− − −	− −	1 0	− −	0 0 −	$B+$
s_{0_2}	∘ ∘ 1	− − 0	− − −	− − −	− −	− −	1 0	0 0 −	C
s_{1_0}	− − −	1 ∘ ∘	− − −	0 − −	1 0	− −	− −	− − −	$A+$
s_{1_1}	− − −	∘ 1 ∘	− 0 −	− − −	− −	1 0	− −	− − −	$B+$
s_{1_2}	− − 0	∘ ∘ 1	− − 0	− − 0	− −	− −	0 1	− − −	C'
s_{2_0}	− − −	− − −	1 ∘ ∘	0 − −	1 0	− −	− −	− − −	$A+$
s_{2_1}	− 0 −	− 0 −	∘ 1 ∘	− 0 −	− −	0 1	− −	− − −	$B'+$
s_{2_2}	− − −	− − 0	∘ ∘ 1	− − −	− −	− −	1 0	− − −	C
s_{3_0}	0 − −	0 − −	0 − −	1 ∘ ∘	0 1	− −	− −	0 0 −	$A'+$
s_{3_1}	− − −	− − −	− 0 −	∘ 1 ∘	− −	1 0	− −	− 0 −	$B+$
s_{3_2}	− − −	− − 0	− − −	∘ ∘ 1	− −	− −	1 0	0 − −	C
s_{4_0}	− − −	− − −	− − −	0 − −	1 ∘	− −	− −	− − −	$A+$
s_{4_1}	0 − −	0 − −	0 − −	− − −	∘ 1	− −	− −	− − −	A'
s_{5_0}	− − −	− − −	− 0 −	− − −	− −	1 ∘	− −	− − −	$B+$
s_{5_1}	− 0 −	− 0 −	− − −	− 0 −	− −	∘ 1	− −	− − −	B'
s_{6_0}	− − −	− − 0	− − −	− − −	− −	− −	1 ∘	− − −	$C+$
s_{6_1}	− − 0	− − −	− − 0	− − 0	− −	− −	∘ 1	− − −	C'
s_{7_0}	− 0 0	− − −	− − −	0 − 0	− −	− −	− −	1 ∘ ∘	$\mathrm{Req}(s_{0_0}) \wedge \mathrm{Req}(s_{3_1})$
s_{7_1}	− 0 0	− − −	− − −	0 0 −	− −	− −	− −	∘ 1 ∘	$\mathrm{Req}(s_{0_0}) \wedge \mathrm{Req}(s_{3_2})$
s_{7_2}	0 − −	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	$s_{0_1} \vee s_{0_2}$

figure 2.10: Identify $\mathrm{MCCR}(r_{0_{0_3}})$ for expansion in $\mathbb{S}_1$.

The expansion of *minor conflict cell rows* $r_{0_{0_3}}$, $r_{0_{1_2}}$, and $r_{0_{2_1}}$ is shown in figure 2.11 at *cells matrix rows* c_7, c_8, and c_9. *Cell* c_{7_8} signifies, that *cell matrix rows* c_7 and c_8 can be *reduced* to a single *cell matrix row* with 5 *state rows*.

P	–––	–––	–––	–––	––	––	––	–––	–––	–––	
s_{0_0}	1 ∘ ∘	–––	–––	0 ––	1 0	––	––	–– 0	0 0 1	0 0 1	
s_{0_1}	∘ 1 ∘	–––	– 0 –	–––	––	1 0	––	0 0 1	–– 0	0 0 1	
s_{0_2}	∘ ∘ 1	–– 0	–––	–––	––	––	1 0	0 0 1	0 0 1	–– 0	
s_{1_0}	–––	1 ∘ ∘	–––	0 ––	1 0	––	––	–––	–––	– 0 –	
s_{1_1}	–––	∘ 1 ∘	– 0 –	–––	––	1 0	––	–––	–––	0 ––	
s_{1_2}	–– 0	∘ ∘ 1	–– 0	–– 0	––	––	0 1	– 0 –	– 0 –	0 0 1	
s_{2_0}	–––	–––	1 ∘ ∘	0 ––	1 0	––	––	–––	– 0 –	–––	
s_{2_1}	– 0 –	– 0 –	∘ 1 ∘	– 0 –	––	0 1	––	0 ––	0 0 1	– 0 –	
s_{2_2}	–––	–– 0	∘ ∘ 1	–––	––	––	1 0	–––	0 ––	–––	
s_{3_0}	0 ––	0 ––	0 ––	1 ∘ ∘	0 1	––	––	0 0 1	0 ––	0 ––	
s_{3_1}	–––	–––	– 0 –	∘ 1 ∘	––	1 0	––	– 0 –	–––	–––	
s_{3_2}	–––	–– 0	–––	∘ ∘ 1	––	––	1 0	0 ––	–––	–––	
s_{4_0}	–––	–––	–––	0 ––	1 ∘	––	––	–––	–––	–––	$A+$
s_{4_1}	0 ––	0 ––	0 ––	–––	∘ 1	––	––	0 0 1	0 ––	0 ––	A'
s_{5_0}	–––	–––	– 0 –	–––	––	1 ∘	––	–––	–––	–––	$B+$
s_{5_1}	– 0 –	– 0 –	–––	– 0 –	––	∘ 1	––	0 ––	0 0 1	– 0 –	B'
s_{6_0}	–––	–– 0	–––	–––	––	––	1 ∘	–––	–––	–––	$C+$
s_{6_1}	–– 0	–––	–– 0	–– 0	––	––	∘ 1	– 0 –	– 0 –	0 0 1	C'
s_{7_0}	1 0 0	–––	– 0 –	0 1 0	1 0	1 0	––	1 ∘ ∘	0 0 1	0 0 1	Mrg(s_{0_0}, s_{3_1})
s_{7_1}	1 0 0	–– 0	–––	0 0 1	1 0	––	1 0	∘ 1 ∘	0 0 1	0 0 1	Mrg(s_{0_0}, s_{3_2})
s_{7_2}	0 ––	–––	–––	–––	––	––	––	∘ ∘ 1	–––	–––	$s_{0_1} \vee s_{0_2}$
s_{8_0}	0 1 0	–––	1 0 0	0 ––	1 0	1 0	––	0 0 1	1 ∘ ∘	0 0 1	MCCR($r_{0_{1_2}}$)
s_{8_1}	0 1 0	–– 0	0 0 1	–––	––	1 0	1 0	0 0 1	∘ 1 ∘	0 0 1	
s_{8_2}	– 0 –	–––	–––	–––	––	––	––	–––	∘ ∘ 1	–––	
s_{9_0}	0 0 1	1 0 0	–––	0 ––	1 0	––	1 0	0 0 1	0 0 1	1 ∘ ∘	MCCR($r_{0_{2_1}}$)
s_{9_1}	0 0 1	0 1 0	– 0 –	–––	––	1 0	1 0	0 0 1	0 0 1	∘ 1 ∘	
s_{9_2}	–– 0	–––	–––	–––	––	––	––	–––	–––	∘ ∘ 1	

figure 2.11: MCCR($r_{0_{0_3}}$), MCCR($r_{0_{1_2}}$), MCCR($r_{0_{2_1}}$) expanded in $\mathbb{S}_1$

In figure 2.12 the relevant *conflict subsequences* of *state rows* s_{10_0} and s_{10_2} are identifed as equal, and therefore *state row* s_{10_2} can be removed. Further, *cells* c_{10_0}, c_{10_7} and c_{10_8} signify that *cell matrix rows* c_0, c_7 and c_8 are *absorbed* and can be removed from $\mathbb{S}_1$.

P	–––	–––	–––	–––	––	––	––	–––	–––	–––	––––––	
s_{0_0}	1 ∘ ∘	–––	–––	0 ––	1 0	––	––	–– 0	0 0 1	0 0 1	–– 0 0 0	
s_{0_1}	∘ 1 ∘	–––	– 0 –	–––	––	1 0	––	0 0 1	–– 0	0 0 1	0 0 –– 0	
s_{0_2}	∘ ∘ 1	–– 0	–––	–––	––	––	1 0	0 0 1	0 0 1	–– 0	0 0 0 0 1	
s_{1_0}	–––	1 ∘ ∘	–––	0 ––	1 0	––	––	–––	–––	– 0 –	––––––	
s_{1_1}	–––	∘ 1 ∘	– 0 –	–––	––	1 0	––	–––	–––	0 ––	––––––	
s_{1_2}	–– 0	∘ ∘ 1	–– 0	–– 0	––	––	0 1	– 0 –	– 0 –	0 0 1	– 0 – 0 0	
s_{2_0}	–––	–––	1 ∘ ∘	0 ––	1 0	––	––	–––	– 0 –	–––	––– 0 –	
s_{2_1}	– 0 –	– 0 –	∘ 1 ∘	– 0 –	––	0 1	––	0 ––	0 0 1	– 0 –	0 – 0 0 –	
s_{2_2}	–––	–– 0	∘ ∘ 1	–––	––	––	1 0	–––	0 ––	–––	–– 0 ––	
s_{3_0}	0 ––	0 ––	0 ––	1 ∘ ∘	0 1	––	––	0 0 1	0 ––	0 ––	0 0 0 ––	
s_{3_1}	–––	–––	– 0 –	∘ 1 ∘	––	1 0	––	– 0 –	–––	–––	– 0 –––	
s_{3_2}	–––	–– 0	–––	∘ ∘ 1	––	––	1 0	0 ––	–––	–––	0 ––––	
s_{4_0}	–––	–––	–––	0 ––	1 ∘	––	––	–––	–––	–––	––––––	$A+$
s_{4_1}	0 ––	0 ––	0 ––	–––	∘ 1	––	––	0 0 1	0 ––	0 ––	0 0 0 ––	A'
s_{5_0}	–––	–––	– 0 –	–––	––	1 ∘	––	–––	–––	–––	––––––	$B+$
s_{5_1}	– 0 –	– 0 –	–––	– 0 –	––	∘ 1	––	0 ––	0 0 1	– 0 –	0 – 0 0 –	B'
s_{6_0}	–––	–– 0	–––	–––	––	––	1 ∘	–––	–––	–––	––––––	$C+$
s_{6_1}	–– 0	–––	–– 0	–– 0	––	––	∘ 1	– 0 –	– 0 –	0 0 1	– 0 – 0 0	C'
s_{7_0}	1 0 0	–––	– 0 –	0 1 0	1 0	1 0	––	1 ∘ ∘	0 0 1	0 0 1	1 0 0 0 0	MCCR($r_{0_{0_3}}$)
s_{7_1}	1 0 0	–– 0	–––	0 0 1	1 0	––	1 0	∘ 1 ∘	0 0 1	0 0 1	0 1 0 0 0	
s_{7_2}	0 ––	–––	–––	–––	––	––	––	∘ ∘ 1	–––	–––	0 0 –––	
s_{8_0}	0 1 0	–––	1 0 0	0 ––	1 0	1 0	––	0 0 1	1 ∘ ∘	0 0 1	0 0 1 0 0	MCCR($r_{0_{1_2}}$)
s_{8_1}	0 1 0	–– 0	0 0 1	–––	––	1 0	1 0	0 0 1	∘ 1 ∘	0 0 1	0 0 0 1 0	
s_{8_2}	– 0 –	–––	–––	–––	––	––	––	–––	∘ ∘ 1	–––	–– 0 0 –	
s_{9_0}	0 0 1	1 0 0	–––	0 ––	1 0	––	1 0	0 0 1	0 0 1	1 ∘ ∘	0 0 0 0 1	MCCR($r_{0_{2_1}}$)
s_{9_1}	0 0 1	0 1 0	– 0 –	–––	––	1 0	1 0	0 0 1	0 0 1	∘ 1 ∘	0 0 0 0 1	
s_{9_2}	–– 0	–––	–––	–––	––	––	––	–––	–––	∘ ∘ 1	––––– 0	
s_{10_0}	1 0 0	–––	– 0 –	0 1 0	1 0	1 0	––	1 0 0	0 0 1	0 0 1	1 ∘ ∘ ∘ ∘	$\subseteq s_{10_2}$
s_{10_1}	1 0 0	–– 0	–––	0 0 1	1 0	––	1 0	0 1 0	0 0 1	0 0 1	∘ 1 ∘ ∘ ∘	
s_{10_2}	0 1 0	–––	1 0 0	0 ––	1 0	1 0	––	0 0 1	1 0 0	0 0 1	∘ ∘ 1 ∘ ∘	$\subseteq s_{10_0} \rightarrow$ kill
s_{10_3}	0 1 0	–– 0	0 0 1	–––	––	1 0	1 0	0 0 1	0 1 0	0 0 1	∘ ∘ ∘ 1 ∘	
s_{10_4}	0 0 1	–– 0	–––	–––	––	––	1 0	0 0 1	0 0 1	–– 0	∘ ∘ ∘ ∘ 1	

figure 2.12: Identify *conflict subsequences* s_{10_0}, s_{10_2} and *absorbed cell matrix rows* c_0, c_7, c_8 in $\mathbb{S}_1$

P	− − −	− − −	− − −	− −	− −	− −	− − −	− − − −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	1 0	− −	− −	− 0 −	− − − −	
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− −	1 0	− −	0 − −	− − − −	
s_{0_2}	∘ ∘ 1	− − 0	− − 0	− −	− −	0 1	0 0 1	− 0 0 0	
s_{1_0}	− − −	1 ∘ ∘	0 − −	1 0	− −	− −	− − −	− − 0 −	
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− −	0 1	− −	− 0 −	0 − 0 −	
s_{1_2}	− − 0	∘ ∘ 1	− − −	− −	− −	1 0	− − −	− − − −	
s_{2_0}	0 − −	0 − −	1 ∘ ∘	0 1	− −	− −	0 − −	0 0 − −	
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− −	1 0	− −	− − −	− 0 − −	
s_{2_2}	− − 0	− − −	∘ ∘ 1	− −	− −	1 0	0 − −	0 − − −	
s_{3_0}	− − −	− − −	0 − −	1 ∘	− −	− −	− − −	− − − −	$A+$
s_{3_1}	0 − −	0 − −	− − −	∘ 1	− −	− −	0 − −	0 0 − −	A'
s_{4_0}	− − −	− 0 −	− − −	− −	1 ∘	− −	− − −	− − − −	$B+$
s_{4_1}	− 0 −	− − −	− 0 −	− −	∘ 1	− −	− 0 −	0 − 0 −	B'
s_{5_0}	− − 0	− − −	− − −	− −	− −	1 ∘	− − −	− − − −	$C+$
s_{5_1}	− − −	− − 0	− − 0	− −	− −	∘ 1	0 0 1	− 0 0 0	C'
s_{6_0}	1 0 0	− − −	0 − −	1 0	− −	1 0	1 ∘ ∘	0 0 0 1	MCCR($r_{0_{2_1}}$)
s_{6_1}	0 1 0	− 0 −	− − −	− −	1 0	1 0	∘ 1 ∘	0 0 0 1	
s_{6_2}	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	− − − 0	
s_{7_0}	− − −	− 0 −	0 1 0	1 0	1 0	− −	0 0 1	1 ∘ ∘ ∘	
s_{7_1}	− − 0	− − −	0 0 1	1 0	− −	1 0	0 0 1	∘ 1 ∘ ∘	
s_{7_2}	− − 0	0 0 1	− − −	− −	1 0	1 0	0 0 1	∘ ∘ 1 ∘	
s_{7_3}	− − 0	− − −	− − −	− −	− −	1 0	− − 0	∘ ∘ ∘ 1	

figure 2.13: bit-counter-s1.n.v-002-merge-sub-red-01.svg

P	− − −	− − −	− − −	− −	− −	− −	− − −	− − − −	− − − − −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	1 0	− −	− −	− 0 −	− − − −	− 0 − − −	$\subseteq s_{0_2}$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− −	1 0	− −	0 − −	− − − −	0 − − − −	$\subseteq s_{0_2}$
s_{0_2}	∘ ∘ 1	1 0 0	0 1 0	1 0	1 0	0 1	0 0 1	1 0 0 0	0 0 1 0 0	$\supset s_{0_0}, \supset s_{0_1} \to$ kill
s_{1_0}	− − −	1 ∘ ∘	0 − −	1 0	− −	− −	− − −	− − 0 −	− − − − 0	$\subseteq s_{1_1}$
s_{1_1}	1 0 0	∘ 1 ∘	0 0 1	1 0	0 1	1 0	− 0 −	0 − 0 −	− 0 0 − 0	$\supset s_{1_0}, \supset s_{1_2} \to$ kill
s_{1_2}	− − 0	∘ ∘ 1	− − −	− −	− −	1 0	− − −	− − − −	− − − − −	$\subseteq s_{1_1}$
s_{2_0}	0 1 0	0 0 1	1 ∘ ∘	0 1	1 0	1 0	0 − −	0 0 − −	0 − 0 0 −	$\supset s_{2_1}, \supset s_{2_2} \to$ kill
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− −	1 0	− −	− − −	− 0 − −	− − − 0 −	$\subseteq s_{2_0}$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− −	− −	1 0	− − −	0 − − −	− − 0 − −	$\subseteq s_{2_0}$
s_{3_0}	− − −	− − −	0 − −	1 ∘	− −	− −	− − −	− − − −	− − − − −	$A+$
s_{3_1}	0 1 0	0 0 1	− − −	∘ 1	1 0	1 0	0 − −	0 0 − −	0 − 0 0 −	A'
s_{4_0}	− − −	− 0 −	− − −	− −	1 ∘	− −	− − −	− − − −	− − − − −	$B+$
s_{4_1}	1 0 0	− − −	0 0 1	1 0	∘ 1	1 0	− 0 −	0 − 0 −	− 0 0 − 0	B'
s_{5_0}	− − 0	− − −	− − −	− −	− −	1 ∘	− − −	− − − −	− − − − −	$C+$
s_{5_1}	− − −	1 0 0	0 1 0	1 0	1 0	∘ 1	0 0 1	1 0 0 0	0 0 1 0 0	C'
s_{6_0}	1 0 0	− − −	0 − −	1 0	− −	1 0	1 ∘ ∘	0 0 0 1	1 0 0 0 0	MCCR($r_{0_{2_1}}$)
s_{6_1}	0 1 0	− 0 −	− − −	− −	1 0	1 0	∘ 1 ∘	0 0 0 1	0 1 0 0 0	
s_{6_2}	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	− − − 0	0 0 − − −	
s_{7_0}	− − −	− 0 −	0 1 0	1 0	1 0	− −	0 0 1	1 ∘ ∘ ∘	0 0 1 0 0	
s_{7_1}	− − 0	− − −	0 0 1	1 0	− −	1 0	0 0 1	∘ 1 ∘ ∘	0 0 0 1 0	
s_{7_2}	− − 0	0 0 1	− − −	− −	1 0	1 0	0 0 1	∘ ∘ 1 ∘	0 0 0 0 1	
s_{7_3}	− − 0	− − −	− − −	− −	− −	1 0	− − 0	∘ ∘ ∘ 1	− − 0 0 0	
s_{8_0}	1 0 0	− − −	0 − −	1 0	− −	1 0	1 0 0	0 0 0 1	1 ∘ ∘ ∘ ∘	$\subseteq s_{8_3}$
s_{8_1}	0 1 0	− 0 −	− − −	− −	1 0	1 0	0 1 0	0 0 0 1	∘ 1 ∘ ∘ ∘	$\subseteq s_{8_4}$
s_{8_2}	− − −	− 0 −	0 1 0	1 0	1 0	− −	0 0 1	1 0 0 0	∘ ∘ 1 ∘ ∘	
s_{8_3}	− − 0	− − −	0 0 1	1 0	− −	1 0	0 0 1	0 1 0 0	∘ ∘ ∘ 1 ∘	$\subseteq s_{8_0} \to$ kill
s_{8_4}	− − 0	0 0 1	− − −	− −	1 0	1 0	0 0 1	0 0 1 0	∘ ∘ ∘ ∘ 1	$\subseteq s_{8_1} \to$ kill

figure 2.14: bit-counter-s1.n.v-002-merge-sub-red-02.svg

P	− −	− −	− −	− −	− −	− −	− − −	
s_{0_0}	1 ∘	− −	− −	1 0	− −	− −	− 0 −	$A+$
s_{0_1}	∘ 1	− −	− −	− −	1 0	− −	0 − −	B
s_{1_0}	− −	1 ∘	− −	1 0	− −	− −	− − −	$A+$
s_{1_1}	− −	∘ 1	− −	− −	− −	1 0	− − −	C
s_{2_0}	− −	− −	1 ∘	− −	1 0	− −	− − −	$B+$
s_{2_1}	− −	− −	∘ 1	− −	− −	1 0	− − 0	C
s_{3_0}	− −	− −	− −	1 ∘	− −	− −	− − −	$A+$
s_{3_1}	0 1	0 1	− −	∘ 1	1 0	1 0	0 − 0	A'
s_{4_0}	− −	− −	− −	− −	1 ∘	− −	− − −	$B+$
s_{4_1}	1 0	− −	0 1	1 0	∘ 1	1 0	− 0 0	B'
s_{5_0}	− −	− −	− −	− −	− −	1 ∘	− − −	$C+$
s_{5_1}	− −	1 0	1 0	1 0	1 0	∘ 1	0 0 1	C'
s_{6_0}	1 0	− −	− −	1 0	− −	1 0	1 ∘ ∘	$AC+$
s_{6_1}	0 1	− −	− −	− −	1 0	1 0	∘ 1 ∘	$BC+$
s_{6_2}	− −	− −	1 0	1 0	1 0	− −	∘ ∘ 1	AB

figure 2.15: bit-counter-s1.n.v-002-merge-sub-red-03.svg

2.4 Alternative Resolution of S_0

P	− − −	− − −	− − −	− − −	− −	− −	− −	− − −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	0 − −	1 0	− −	− −	− − 0	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− 0 −	− −	1 0	− −	0 0 −	$B+$
s_{0_2}	∘ ∘ 1	− − 0	− − 0	− − −	− −	− −	1 0	0 0 −	C
s_{1_0}	− − −	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	− − −	$A+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	− − −	$B'+$
s_{1_2}	− − 0	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	− − −	C'
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	0 0 −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	− 0 −	$B+$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 − −	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	− − −	$A'+$
s_{3_1}	− 0 −	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	− − −	$B'+$
s_{3_2}	− − −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− − −	C
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	− − −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	− − −	A'
s_{5_0}	− − −	− 0 −	− − −	− 0 −	− −	1 ∘	− −	− − −	$B+$
s_{5_1}	− 0 −	− − −	− 0 −	− − −	− −	∘ 1	− −	− − −	B'
s_{6_0}	− − −	− − 0	− − 0	− − −	− −	− −	1 ∘	− − −	$C+$
s_{6_1}	− − 0	− − −	− − −	− − 0	− −	− −	∘ 1	− − −	C'
s_{7_0}	− 0 0	− − −	0 − 0	− − −	− −	− −	− −	1 ∘ ∘	MCCR($r_{0_{0_2}}$)
s_{7_1}	− 0 0	− − −	0 0 −	− − −	− −	− −	− −	∘ 1 ∘	
s_{7_2}	0 − −	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	

figure 2.16: bit-counter-s0.n.v-010-mmcr-00.svg

P	− − −	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	− − −	− − −	− − −	− − −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	0 − −	1 0	− −	− −	− − 0	− − 0	0 0 1	0 0 1	0 0 1	0 0 1	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− 0 −	− −	1 0	− −	0 0 1	0 0 1	− − 0	− − 0	0 0 1	0 0 1	$B+$
s_{0_2}	∘ ∘ 1	− − 0	− − 0	− − −	− −	− −	1 0	0 0 1	0 0 1	0 0 1	0 0 1	− − 0	− − 0	C
s_{1_0}	− − −	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	− − −	− − −	− 0 −	0 − −	− 0 −	0 − −	$A+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	− 0 −	0 − −	0 0 1	0 0 1	0 − −	− 0 −	$B'+$
s_{1_2}	− − 0	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	− 0 −	0 − −	0 − −	− 0 −	0 0 1	0 0 1	C'
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	0 0 1	0 0 1	0 − −	− 0 −	0 − −	− 0 −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	0 − −	− 0 −	− − −	− − −	− 0 −	0 − −	$B+$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	− 0 −	0 − −	0 − −	− 0 −	0 0 1	0 0 1	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	0 0 1	0 0 1	0 − −	− 0 −	0 − −	− 0 −	$A'+$
s_{3_1}	− 0 −	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	− 0 −	0 − −	0 0 1	0 0 1	0 − −	− 0 −	$B'+$
s_{3_2}	− − −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	0 − −	− 0 −	− 0 −	0 − −	− − −	− − −	C
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	− − −	− − −	− 0 −	0 − −	− 0 −	0 − −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	0 0 1	0 0 1	0 − −	− 0 −	0 − −	− 0 −	A'
s_{5_0}	− − −	− 0 −	− − −	− 0 −	− −	1 ∘	− −	0 − −	− 0 −	− − −	− − −	− 0 −	0 − −	$B+$
s_{5_1}	− 0 −	− − −	− 0 −	− − −	− −	∘ 1	− −	− 0 −	0 − −	0 0 1	0 0 1	0 − −	− 0 −	B'
s_{6_0}	− − −	− − 0	− − 0	− − −	− −	− −	1 ∘	0 − −	− 0 −	− 0 −	0 − −	− − −	− − −	$C+$
s_{6_1}	− − 0	− − −	− − −	− − 0	− −	− −	∘ 1	− 0 −	0 − −	0 − −	− 0 −	0 0 1	0 0 1	C'
s_{7_0}	1 0 0	− − −	0 0 1	0 1 0	1 0	0 1	0 1	1 ∘ ∘	0 1 0	0 0 1	0 0 1	0 0 1	0 0 1	MCCR($r_{0_{0_2}}$)
s_{7_1}	1 0 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘	1 0 0	0 0 1	0 0 1	0 0 1	0 0 1	
s_{7_2}	0 − −	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	0 0 1	− − −	− − −	− − −	− − −	
s_{8_0}	1 0 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 1 0	1 ∘ ∘	0 0 1	0 0 1	0 0 1	0 0 1	MCCR($r_{0_{0_3}}$)
s_{8_1}	1 0 0	− − −	0 0 1	0 1 0	1 0	0 1	0 1	1 0 0	∘ 1 ∘	0 0 1	0 0 1	0 0 1	0 0 1	Abs(c_{7_8}) → kill
s_{8_2}	0 − −	− − −	− − −	− − −	− −	− −	− −	0 0 1	∘ ∘ 1	− − −	− − −	− − −	− − −	
s_{9_0}	0 1 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 0 1	1 ∘ ∘	0 1 0	0 0 1	0 0 1	MCCR($r_{0_{1_1}}$)
s_{9_1}	0 1 0	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	0 0 1	∘ 1 ∘	1 0 0	0 0 1	0 0 1	
s_{9_2}	− 0 −	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	∘ ∘ 1	0 0 1	− − −	− − −	
s_{10_0}	0 1 0	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	0 0 1	0 1 0	1 ∘ ∘	0 0 1	0 0 1	MCCR($r_{0_{1_3}}$)
s_{10_1}	0 1 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 0 1	1 0 0	∘ 1 ∘	0 0 1	0 0 1	Abs($c_{9_{10}}$) → kill
s_{10_2}	− 0 −	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	0 0 1	∘ ∘ 1	− − −	− − −	
s_{11_0}	0 0 1	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 0 1	0 0 1	0 0 1	1 ∘ ∘	0 1 0	MCCR($r_{0_{2_1}}$)
s_{11_1}	0 0 1	0 1 0	1 0 0	− − −	0 1	0 1	1 0	0 0 1	0 0 1	0 0 1	0 0 1	∘ 1 ∘	1 0 0	
s_{11_2}	− − 0	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	− − −	− − −	∘ ∘ 1	0 0 1	
s_{12_0}	0 0 1	0 1 0	1 0 0	− − −	0 1	0 1	1 0	0 0 1	0 0 1	0 0 1	0 0 1	0 1 0	1 ∘ ∘	MCCR($r_{0_{2_2}}$)
s_{12_1}	0 0 1	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 0 1	0 0 1	0 0 1	1 0 0	∘ 1 ∘	Abs($c_{11_{12}}$) → kill
s_{12_2}	− − 0	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	− − −	− − −	0 0 1	∘ ∘ 1	

figure 2.17: bit-counter-s0.n.v-010-mmcr-01.svg

P	− − −	− − −	− − −	− − −	− − −	− −	− −	− − −	− − −	− − −	− − − − −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	0 − −	1 0	− −	− −	− − 0	0 0 1	0 0 1	− − − − −	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− 0 −	− −	1 0	− −	0 0 1	− − 0	0 0 1	− − − − −	$B+$
s_{0_2}	∘ ∘ 1	− − 0	− − 0	− − −	− −	− −	1 0	0 0 1	0 0 1	− − 0	− − − − −	C
s_{1_0}	− − −	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	− − −	0 − −	0 − −	− − − − −	$A+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	0 − −	0 0 1	− 0 −	− − − − −	$B'+$
s_{1_2}	− − 0	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	0 − −	− 0 −	0 0 1	− − − − −	C'
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	0 0 1	− 0 −	− 0 −	− − − − −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	− 0 −	− − −	0 − −	− − − − −	$B+$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 − −	− 0 −	0 0 1	− − − − −	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	0 0 1	− 0 −	− 0 −	− − − − −	$A'+$
s_{3_1}	− 0 −	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	0 − −	0 0 1	− 0 −	− − − − −	$B'+$
s_{3_2}	− − −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− 0 −	0 − −	− − −	− − − − −	C
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	− − −	0 − −	0 − −	− − 0 − −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	0 0 1	− 0 −	− 0 −	0 0 − 0 −	A'
s_{5_0}	− − −	− 0 −	− − −	− 0 −	− −	1 ∘	− −	− 0 −	− − −	0 − −	− 0 − − −	$B+$
s_{5_1}	− 0 −	− − −	− 0 −	− − −	− −	∘ 1	− −	0 − −	0 0 1	− 0 −	0 − 0 0 −	B'
s_{6_0}	− − −	− − 0	− − 0	− − −	− −	− −	1 ∘	− 0 −	0 − −	− − −	− 0 0 − −	$C+$
s_{6_1}	− − 0	− − −	− − −	− − 0	− −	− −	∘ 1	0 − −	− 0 −	0 0 1	0 − − 0 0	C'
s_{7_0}	1 0 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	1 ∘ ∘	0 0 1	0 0 1	− 0 0 0 0	MCCR($r_{0_{0_2}}$)
s_{7_1}	1 0 0	− − −	0 0 1	0 1 0	1 0	0 1	0 1	∘ 1 ∘	0 0 1	0 0 1	0 − 0 0 0	MCCR($r_{0_{0_3}}$)
s_{7_2}	0 − −	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	− − −	− − −	0 0 − − −	
s_{8_0}	0 1 0	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	1 ∘ ∘	0 0 1	0 0 − 0 0	MCCR($r_{0_{1_1}}$)
s_{8_1}	0 1 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	∘ 1 ∘	0 0 1	0 0 0 − 0	MCCR($r_{0_{1_3}}$)
s_{8_2}	− 0 −	− − −	− − −	− − −	− −	− −	− −	− − −	∘ ∘ 1	− − −	− − 0 0 −	
s_{9_0}	0 0 1	0 1 0	1 0 0	− − −	0 1	0 1	1 0	0 0 1	0 0 1	1 ∘ ∘	− − − − −	MCCR($r_{0_{2_1}}$)
s_{9_1}	0 0 1	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 0 1	∘ 1 ∘	− − − − −	MCCR($r_{0_{2_2}}$)
s_{9_2}	− − 0	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	∘ ∘ 1	− − − − −	
s_{10_0}	− − −	− − −	− − −	− − −	− −	− −	− −	− 0 0	0 0 −	− − −	1 ∘ ∘ ∘ ∘	Mrg(c_7, c_8)
s_{10_1}	− − −	− − −	− − −	− − −	− −	− −	− −	0 − 0	0 0 −	− − −	∘ 1 ∘ ∘ ∘	
s_{10_2}	− − −	− − −	− − −	− − −	− −	− −	− −	0 0 −	− 0 0	− − −	∘ ∘ 1 ∘ ∘	
s_{10_3}	− − −	− − −	− − −	− − −	− −	− −	− −	0 0 −	0 − 0	− − −	∘ ∘ ∘ 1 ∘	
s_{10_4}	− − −	− − −	− − −	− − −	− −	− −	− −	0 0 −	0 0 −	− − −	∘ ∘ ∘ ∘ 1	

figure 2.18: bit-counter-s0.n.v-010-mmcr-02.svg

P	− − −	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	− − −	− − − − −	
s_{0_0}	1 ∘ ∘	− − −	0 − −	0 − −	1 0	− −	− −	− − 0	0 0 1	0 0 1	− − 0 0 0	Abs(r_{10_0}) → kill
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− 0 −	− −	1 0	− −	0 0 1	− − 0	0 0 1	0 0 − − 0	
s_{0_2}	∘ ∘ 1	− − 0	− − 0	− − −	− −	− −	1 0	0 0 1	0 0 1	− − 0	0 0 0 0 1	
s_{1_0}	− − −	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	− − −	0 − −	0 − −	− − 0 − −	$A+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	0 − −	0 0 1	− 0 −	0 − 0 0 −	$B'+$
s_{1_2}	− − 0	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	0 − −	− 0 −	0 0 1	0 − − 0 0	C'
s_{2_0}	0 − −	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	0 0 1	− 0 −	− 0 −	0 0 − 0 −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	− 0 −	− − −	0 − −	− 0 − − −	$B+$
s_{2_2}	− − 0	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 − −	− 0 −	0 0 1	0 − − 0 0	C'
s_{3_0}	0 − −	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	0 0 1	− 0 −	− 0 −	0 0 − 0 −	$A'+$
s_{3_1}	− 0 −	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	0 − −	0 0 1	− 0 −	0 − 0 0 −	$B'+$
s_{3_2}	− − −	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− 0 −	0 − −	− − −	− 0 0 − −	C
s_{4_0}	− − −	− − −	0 − −	0 − −	1 ∘	− −	− −	− − −	0 − −	0 − −	− − 0 − −	$A+$
s_{4_1}	0 − −	0 − −	− − −	− − −	∘ 1	− −	− −	0 0 1	− 0 −	− 0 −	0 0 − 0 −	A'
s_{5_0}	− − −	− 0 −	− − −	− 0 −	− −	1 ∘	− −	− 0 −	− − −	0 − −	− 0 − − −	$B+$
s_{5_1}	− 0 −	− − −	− 0 −	− − −	− −	∘ 1	− −	0 − −	0 0 1	− 0 −	0 − 0 0 −	B'
s_{6_0}	− − −	− − 0	− − 0	− − −	− −	− −	1 ∘	− 0 −	0 − −	− − −	− 0 0 − −	$C+$
s_{6_1}	− − 0	− − −	− − −	− − 0	− −	− −	∘ 1	0 − −	− 0 −	0 0 1	0 − − 0 0	C'
s_{7_0}	1 0 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	1 ∘ ∘	0 0 1	0 0 1	1 0 0 0 0	MCCR($r_{0_{0_2}}$)
s_{7_1}	1 0 0	− − −	0 0 1	0 1 0	1 0	0 1	0 1	∘ 1 ∘	0 0 1	0 0 1	0 1 0 0 0	MCCR($r_{0_{0_3}}$)
s_{7_2}	0 − −	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	− − −	− − −	0 0 − − −	Abs(c_{10_7}) → kill
s_{8_0}	0 1 0	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	1 ∘ ∘	0 0 1	0 0 1 0 0	MCCR($r_{0_{1_1}}$)
s_{8_1}	0 1 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	∘ 1 ∘	0 0 1	0 0 0 1 0	MCCR($r_{0_{1_3}}$)
s_{8_2}	− 0 −	− − −	− − −	− − −	− −	− −	− −	− − −	∘ ∘ 1	− − −	− − 0 0 −	Abs(c_{10_8}) → kill
s_{9_0}	0 0 1	0 1 0	1 0 0	− − −	0 1	0 1	1 0	0 0 1	0 0 1	1 ∘ ∘	0 0 0 0 1	MCCR($r_{0_{2_1}}$)
s_{9_1}	0 0 1	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 0 1	∘ 1 ∘	0 0 0 0 1	MCCR($r_{0_{2_2}}$)
s_{9_2}	− − 0	− − −	− − −	− − −	− −	− −	− −	− − −	− − −	∘ ∘ 1	− − − − 0	
s_{10_0}	1 0 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	1 0 0	0 0 1	0 0 1	1 ∘ ∘ ∘ ∘	
s_{10_1}	1 0 0	− − −	0 0 1	0 1 0	1 0	0 1	0 1	0 1 0	0 0 1	0 0 1	∘ 1 ∘ ∘ ∘	
s_{10_2}	0 1 0	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	1 0 0	0 0 1	∘ ∘ 1 ∘ ∘	
s_{10_3}	0 1 0	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	0 1 0	0 0 1	∘ ∘ ∘ 1 ∘	$\supseteq s_{10_0} \rightarrow$ kill
s_{10_4}	0 0 1	− − 0	− − 0	− − −	− −	− −	1 0	0 0 1	0 0 1	− − 0	∘ ∘ ∘ ∘ 1	

figure 2.19: bit-counter-s0.n.v-010-mmcr-03.svg

P	− − −	− − −	− − −	− −	− −	− −	− − −	− − − −	
s_{0_0}	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	0 − −	− − 0 −	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	− 0 −	0 − 0 −	$B'+$
s_{0_2}	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	0 0 1	0 − − 0	C'
s_{1_0}	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	− 0 −	0 0 − −	$A'+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	0 − −	− 0 − −	$B+$
s_{1_2}	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 0 1	0 − − 0	C'
s_{2_0}	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	− 0 −	0 0 − −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	− 0 −	0 − 0 −	$B'+$
s_{2_2}	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− − −	− 0 0 −	C
s_{3_0}	− − −	0 − −	0 − −	1 ∘	− −	− −	0 − −	− − 0 −	$A+$
s_{3_1}	0 − −	− − −	− − −	∘ 1	− −	− −	− 0 −	0 0 − −	A'
s_{4_0}	− 0 −	− − −	− 0 −	− −	1 ∘	− −	0 − −	− 0 − −	$B+$
s_{4_1}	− − −	− 0 −	− − −	− −	∘ 1	− −	− 0 −	0 − 0 −	B'
s_{5_0}	− − 0	− − 0	− − −	− −	− −	1 ∘	− − −	− 0 0 −	$C+$
s_{5_1}	− − −	− − −	− − 0	− −	− −	∘ 1	0 0 1	0 − − 0	C'
s_{6_0}	0 1 0	1 0 0	− − −	0 1	0 1	1 0	1 ∘ ∘	0 0 0 1	MCCR($r_{0_{2_1}}$)
s_{6_1}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘	0 0 0 1	MCCR($r_{0_{2_2}}$)
s_{6_2}	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	− − − 0	
s_{7_0}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	1 ∘ ∘ ∘	
s_{7_1}	− − −	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1	∘ 1 ∘ ∘	
s_{7_2}	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	∘ ∘ 1 ∘	
s_{7_3}	− − 0	− − 0	− − −	− −	− −	1 0	− − 0	∘ ∘ ∘ 1	

figure 2.20: bit-counter-s0.n.v-010-mmcr-04.svg

P	− − −	− − −	− − −	− −	− −	− −	− − −	− − − −	− − − − −	
s_{0_0}	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	0 − −	− − 0 −	0 − − − 0	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	− 0 −	0 − 0 −	− 0 0 − 0	$B'+$
s_{0_2}	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	0 0 1	0 − − 0	0 0 0 − −	C'
s_{1_0}	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	− 0 −	0 0 − −	− 0 0 0 −	$A'+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	0 − −	− 0 − −	0 − − 0 −	$B+$
s_{1_2}	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 0 1	0 − − 0	0 0 0 − −	C'
s_{2_0}	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	− 0 −	0 0 − −	− 0 0 0 −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	− 0 −	0 − 0 −	− 0 0 − 0	$B'+$
s_{2_2}	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− − −	− 0 0 −	− − − 0 0	C
s_{3_0}	− − −	0 − −	0 − −	1 ∘	− −	− −	0 − −	− − 0 −	0 − − − 0	$A+$
s_{3_1}	0 − −	− − −	− − −	∘ 1	− −	− −	− 0 −	0 0 − −	− 0 0 0 −	A'
s_{4_0}	− 0 −	− − −	− 0 −	− −	1 ∘	− −	0 − −	− 0 − −	0 − − 0 −	$B+$
s_{4_1}	− − −	− 0 −	− − −	− −	∘ 1	− −	− 0 −	0 − 0 −	− 0 0 − 0	B'
s_{5_0}	− − 0	− − 0	− − −	− −	− −	1 ∘	− − −	− 0 0 −	− − − 0 0	$C+$
s_{5_1}	− − −	− − −	− − 0	− −	− −	∘ 1	0 0 1	0 − − 0	0 0 0 − −	C'
s_{6_0}	0 1 0	1 0 0	− − −	0 1	0 1	1 0	1 ∘ ∘	0 0 0 1	1 0 0 0 0	MCCR($r_{0_{2_1}}$)
s_{6_1}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘	0 0 0 1	0 1 0 0 0	MCCR($r_{0_{2_2}}$)
s_{6_2}	− − −	− − −	− − −	− −	− −	− −	∘ ∘ 1	− − − 0	0 0 − − −	Abs(c_{8_6}) → kill
s_{7_0}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	1 ∘ ∘ ∘	0 0 1 0 0	Abs(c_{8_7}) → kill
s_{7_1}	− − −	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1	∘ 1 ∘ ∘	0 0 0 1 0	
s_{7_2}	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	∘ ∘ 1 ∘	0 0 0 0 1	
s_{7_3}	− − 0	− − 0	− − −	− −	− −	1 0	− − 0	∘ ∘ ∘ 1	− − 0 0 0	
s_{8_0}	0 1 0	1 0 0	− − −	0 1	0 1	1 0	1 0 0	0 0 0 1	1 ∘ ∘ ∘ ∘	
s_{8_1}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 1 0	0 0 0 1	∘ 1 ∘ ∘ ∘	
s_{8_2}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	0 0 1	1 0 0 0	∘ ∘ 1 ∘ ∘	$\supseteq s_{8_1}$ → kill
s_{8_3}	− − −	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1	0 1 0 0	∘ ∘ ∘ 1 ∘	
s_{8_4}	0 0 1	− − −	1 0 0	0 1	1 0	0 1	0 0 1	0 0 1 0	∘ ∘ ∘ ∘ 1	

figure 2.21: bit-counter-s0.n.v-010-mmcr-05.svg

P	− − −	− − −	− − −	− −	− −	− −	− − − −	
s_{0_0}	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	0 − − 0	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	− 0 − 0	$B'+$
s_{0_2}	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	0 0 − −	C'
s_{1_0}	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	− 0 0 −	$A'+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	0 − 0 −	$B+$
s_{1_2}	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 0 − −	C'
s_{2_0}	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	− 0 0 −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	− 0 − 0	$B'+$
s_{2_2}	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− − 0 0	C
s_{3_0}	− − −	0 − −	0 − −	1 ∘	− −	− −	0 − − 0	$A+$
s_{3_1}	0 − −	− − −	− − −	∘ 1	− −	− −	− 0 0 −	A'
s_{4_0}	− 0 −	− − −	− 0 −	− −	1 ∘	− −	0 − 0 −	$B+$
s_{4_1}	− − −	− 0 −	− − −	− −	∘ 1	− −	− 0 − 0	B'
s_{5_0}	− − 0	− − 0	− − −	− −	− −	1 ∘	− − 0 0	$C+$
s_{5_1}	− − −	− − −	− − 0	− −	− −	∘ 1	0 0 − −	C'
s_{6_0}	0 1 0	1 0 0	− − −	0 1	0 1	1 0	1 ∘ ∘ ∘	
s_{6_1}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘ ∘	
s_{6_2}	− − −	0 0 1	0 1 0	1 0	0 1	0 1	∘ ∘ 1 ∘	
s_{6_3}	0 0 1	− − −	1 0 0	0 1	1 0	0 1	∘ ∘ ∘ 1	

figure 2.22: bit-counter-s0.n.v-010-mmcr-06.svg

P	− − −	− − −	− − −	− −	− −	− −	− − − −	− − − −	− − − −	− − − −	
s_{0_0}	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	0 − − 0	− 0 0 −	0 0 0 1	0 0 0 1	$A+$
s_{0_1}	∘ 1 ∘	− 0 −	− − −	− −	0 1	− −	− 0 − 0	0 − 0 −	0 0 0 1	− − − −	$B'+$
s_{0_2}	∘ ∘ 1	− − −	− − 0	− −	− −	0 1	0 0 − −	0 0 − −	− − − −	0 0 0 1	C'
s_{1_0}	0 − −	1 ∘ ∘	− − −	0 1	− −	− −	− 0 0 −	0 0 0 1	− 0 0 −	− − − −	$A'+$
s_{1_1}	− 0 −	∘ 1 ∘	− 0 −	− −	1 0	− −	0 − 0 −	0 0 0 1	0 − 0 −	0 0 0 1	$B+$
s_{1_2}	− − −	∘ ∘ 1	− − 0	− −	− −	0 1	0 0 − −	− − − −	0 0 − −	0 0 0 1	C'
s_{2_0}	0 − −	− − −	1 ∘ ∘	0 1	− −	− −	− 0 0 −	0 0 0 1	− − − −	− 0 0 −	$A'+$
s_{2_1}	− − −	− 0 −	∘ 1 ∘	− −	0 1	− −	− 0 − 0	− − − −	0 0 0 1	0 − 0 −	$B'+$
s_{2_2}	− − 0	− − 0	∘ ∘ 1	− −	− −	1 0	− − 0 0	0 0 0 1	0 0 0 1	0 0 − −	C
s_{3_0}	− − −	0 − −	0 − −	1 ∘	− −	− −	0 − − 0	− − − −	0 0 0 1	0 0 0 1	$A+$
s_{3_1}	0 − −	− − −	− − −	∘ 1	− −	− −	− 0 0 −	0 0 0 1	− − − −	− − − −	A'
s_{4_0}	− 0 −	− − −	− 0 −	− −	1 ∘	− −	0 − 0 −	0 0 0 1	− − − −	0 0 0 1	$B+$
s_{4_1}	− − −	− 0 −	− − −	− −	∘ 1	− −	− 0 − 0	− − − −	0 0 0 1	− − − −	B'
s_{5_0}	− − 0	− − 0	− − −	− −	− −	1 ∘	− − 0 0	0 0 0 1	0 0 0 1	− − − −	$C+$
s_{5_1}	− − −	− − −	− − 0	− −	− −	∘ 1	0 0 − −	− − − −	− − − −	0 0 0 1	C'
s_{6_0}	0 1 0	1 0 0	− − −	0 1	0 1	1 0	1 ∘ ∘ ∘	0 0 0 1	0 0 0 1	− − − 0	
s_{6_1}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘ ∘	0 0 0 1	0 0 0 1	0 0 0 1	
s_{6_2}	− − −	0 0 1	0 1 0	1 0	0 1	0 1	∘ ∘ 1 ∘	− − − 0	0 0 0 1	0 0 0 1	
s_{6_3}	0 0 1	− − −	1 0 0	0 1	1 0	0 1	∘ ∘ ∘ 1	0 0 0 1	− − − 0	0 0 0 1	
s_{7_0}	1 0 0	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1 0	1 ∘ ∘ ∘	0 0 0 1	0 0 0 1	$\mathrm{Mrg}(s_{6_2}, c_0)$
s_{7_1}	0 1 0	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1 0	∘ 1 ∘ ∘	0 0 0 1	0 0 0 1	$\supseteq s_{7_0} \to$ kill
s_{7_2}	0 0 1	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1 0	∘ ∘ 1 ∘	0 0 0 1	0 0 0 1	$\supseteq s_{7_0} \to$ kill
s_{7_3}	− − −	− − −	− − −	− −	− −	− −	− − 0 −	∘ ∘ ∘ 1	− − − −	− − − −	
s_{8_0}	0 0 1	1 0 0	1 0 0	0 1	1 0	0 1	0 0 0 1	0 0 0 1	1 ∘ ∘ ∘	0 0 0 1	$\mathrm{Mrg}(s_{6_3}, c_1)$
s_{8_1}	0 0 1	0 1 0	1 0 0	0 1	1 0	0 1	0 0 0 1	0 0 0 1	∘ 1 ∘ ∘	0 0 0 1	$\supseteq s_{8_0} \to$ kill
s_{8_2}	0 0 1	0 0 1	1 0 0	0 1	1 0	0 1	0 0 0 1	0 0 0 1	∘ ∘ 1 ∘	0 0 0 1	$\supseteq s_{8_0} \to$ kill
s_{8_3}	− − −	− − −	− − −	− −	− −	− −	− − − 0	− − − −	∘ ∘ ∘ 1	− − − −	
s_{9_0}	0 1 0	1 0 0	1 0 0	0 1	0 1	1 0	1 0 0 0	0 0 0 1	0 0 0 1	1 ∘ ∘ ∘	$\mathrm{Mrg}(s_{6_0}, c_2)$
s_{9_1}	0 1 0	1 0 0	0 1 0	0 1	0 1	1 0	1 0 0 0	0 0 0 1	0 0 0 1	∘ 1 ∘ ∘	$\supseteq s_{9_0} \to$ kill
s_{9_2}	0 1 0	1 0 0	0 0 1	0 1	0 1	1 0	1 0 0 0	0 0 0 1	0 0 0 1	∘ ∘ 1 ∘	$\supseteq s_{9_0} \to$ kill
s_{9_3}	− − −	− − −	− − −	− −	− −	− −	0 − − −	− − − −	− − − −	∘ ∘ ∘ 1	

figure 2.23: bit-counter-s0.n.v-010-mmcr-07.svg

P	− − −	− − −	− − −	− −	− −	− −	− − − −	− −	− −	− −	
s_{0_0}	1 ∘ ∘	0 − −	0 − −	1 0	− −	− −	0 − − 0	− −	0 1	0 1	$A+$
s_{0_1}	∘ 1 ∘	1 0 0	1 0 0	0 1	0 1	1 0	1 0 0 0	0 1	0 1	1 0	$A'B'C+$
s_{0_2}	∘ ∘ 1	1 0 0	1 0 0	0 1	1 0	0 1	0 0 0 1	0 1	1 0	0 1	$A'BC'$
s_{1_0}	0 − −	1 ∘ ∘	1 0 0	0 1	− −	− −	− 0 0 −	0 1	− −	− −	$\mathrm{Abs}(c_{2_1}) \to \mathrm{kill}$
s_{1_1}	1 0 0	∘ 1 ∘	0 0 1	1 0	1 0	1 0	0 1 0 0	0 1	0 1	0 1	
s_{1_2}	1 0 0	∘ ∘ 1	0 1 0	1 0	0 1	0 1	0 0 1 0	1 0	0 1	0 1	
s_{2_0}	0 − −	1 0 0	1 ∘ ∘	0 1	− −	− −	− 0 0 −	0 1	− −	− −	$A'+$
s_{2_1}	1 0 0	0 0 1	∘ 1 ∘	1 0	0 1	0 1	0 0 1 0	1 0	0 1	0 1	$AB'C'+$
s_{2_2}	1 0 0	0 1 0	∘ ∘ 1	1 0	1 0	1 0	0 1 0 0	0 1	0 1	0 1	ABC
s_{3_0}	1 0 0	0 − −	0 − −	1 ∘	− −	− −	0 − − 0	− −	0 1	0 1	$A+$
s_{3_1}	0 − −	1 0 0	1 0 0	∘ 1	− −	− −	− 0 0 −	0 1	− −	− −	A'
s_{4_0}	− 0 −	− − 0	− 0 −	− −	1 ∘	− −	0 − 0 −	0 1	− −	0 1	$B+$
s_{4_1}	− − 0	− 0 −	− − 0	− −	∘ 1	− −	− 0 − 0	− −	0 1	− −	B'
s_{5_0}	− − 0	− − 0	− 0 −	− −	− −	1 ∘	− − 0 0	0 1	0 1	− −	$C+$
s_{5_1}	− 0 −	− 0 −	− − 0	− −	− −	∘ 1	0 0 − −	− −	− −	0 1	C'
s_{6_0}	0 1 0	1 0 0	1 0 0	0 1	0 1	1 0	1 ∘ ∘ ∘	0 1	0 1	1 0	$A'B'C+$
s_{6_1}	1 0 0	0 1 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘ ∘	0 1	0 1	0 1	ABC
s_{6_2}	1 0 0	0 0 1	0 1 0	1 0	0 1	0 1	∘ ∘ 1 ∘	1 0	0 1	0 1	$AB'C'+$
s_{6_3}	0 0 1	1 0 0	1 0 0	0 1	1 0	0 1	∘ ∘ ∘ 1	0 1	1 0	0 1	$A'BC'$
s_{7_0}	1 0 0	0 0 1	0 1 0	1 0	0 1	0 1	0 0 1 0	1 ∘	0 1	0 1	$\mathrm{Abs}(c_{6_7}) \to \mathrm{kill}$
s_{7_1}	− − −	− − 0	− 0 −	− −	− −	− −	− − 0 −	∘ 1	− −	− −	
s_{8_0}	0 0 1	1 0 0	1 0 0	0 1	1 0	0 1	0 0 0 1	0 1	1 ∘	0 1	$\mathrm{Abs}(c_{6_8}) \to \mathrm{kill}$
s_{8_1}	− − 0	− − −	− − −	− −	− −	− −	− − − 0	− −	∘ 1	− −	
s_{9_0}	0 1 0	1 0 0	1 0 0	0 1	0 1	1 0	1 0 0 0	0 1	0 1	1 ∘	$\mathrm{Abs}(c_{6_9}) \to \mathrm{kill}$
s_{9_1}	− 0 −	− − −	− − −	− −	− −	− −	0 − − −	− −	− −	∘ 1	

figure 2.24: bit-counter-s0.n.v-010-mmcr-08.svg

P	− − −	− − −	− −	− −	− −	− − − −	
s_{0_0}	1 ∘ ∘	0 − −	1 0	− −	− −	0 − − 0	$A+$
s_{0_1}	∘ 1 ∘	1 0 0	0 1	0 1	1 0	1 0 0 0	$A'B'C+$
s_{0_2}	∘ ∘ 1	1 0 0	0 1	1 0	0 1	0 0 0 1	$A'BC'$
s_{1_0}	0 − −	1 ∘ ∘	0 1	− −	− −	− 0 0 −	$A'+$
s_{1_1}	1 0 0	∘ 1 ∘	1 0	0 1	0 1	0 0 1 0	$AB'C'+$
s_{1_2}	1 0 0	∘ ∘ 1	1 0	1 0	1 0	0 1 0 0	ABC
s_{2_0}	1 0 0	0 − −	1 ∘	− −	− −	0 − − 0	$A+$
s_{2_1}	0 − −	1 0 0	∘ 1	− −	− −	− 0 0 −	A'
s_{3_0}	− 0 −	− 0 −	− −	1 ∘	− −	0 − 0 −	$B+$
s_{3_1}	− − 0	− − 0	− −	∘ 1	− −	− 0 − 0	B'
s_{4_0}	− − 0	− 0 −	− −	− −	1 ∘	− − 0 0	$C+$
s_{4_1}	− 0 −	− − 0	− −	− −	∘ 1	0 0 − −	C'
s_{5_0}	0 1 0	1 0 0	0 1	0 1	1 0	1 ∘ ∘ ∘	$A'B'C+$
s_{5_1}	1 0 0	0 0 1	1 0	1 0	1 0	∘ 1 ∘ ∘	ABC
s_{5_2}	1 0 0	0 1 0	1 0	0 1	0 1	∘ ∘ 1 ∘	$AB'C'+$
s_{5_3}	0 0 1	1 0 0	0 1	1 0	0 1	∘ ∘ ∘ 1	$A'BC'$

figure 2.25: bit-counter-s0.n.v-010-mmcr-09.svg

ABBREVIATIONS

BCF see *Blake Canonical Form*

CNF see *Conjunctive normal form*

DNF see *Disjunctive normal form*

DPLL see *DPLL algorithm*

GLOSSARY

Adjacency matrix As Wikipedia describes it[WPADJ]:

> In graph theory and computer science, an adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices are adjacent or not in the graph.
>
> In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix with zeros on its diagonal. If the graph is undirected (i.e. all of its edges are bidirectional), the adjacency matrix is symmetric.

Blake Canonical Form As Wikipedia describes it[WPBCF]:

> In Boolean logic, a formula for a Boolean function f is in Blake canonical form (BCF), also called the complete sum of prime implicants, the complete sum, or the disjunctive prime form, when it is a disjunction of all the prime implicants of f. [...]
>
> The Blake canonical form is a special case of disjunctive normal form.
>
> The Blake canonical form is not necessarily minimal, however all the terms of a minimal sum are contained in the Blake canonical form. [...]
>
> Selecting a minimal sum from a Blake canonical form amounts in general to solving the set cover problem, so is NP-hard.

Conjunctive normal form As Wikipedia describes it[WPCNF]:

> In Boolean algebra, a formula is in conjunctive normal form (CNF) or clausal normal form if it is a conjunction of one or more clauses, where a clause is a disjunction of literals; otherwise put, it is a product of sums or an AND of ORs.
>
> A logical formula is considered to be in CNF if it is a conjunction of one or more disjunctions of one or more literals. As in disjunctive normal form (DNF), the only propositional operators in CNF are or ($\vee$), and ($\wedge$), and not ($\neg$). The not operator can only be used as part of a literal, which means that it can only precede a propositional variable.

Disjunctive normal form As Wikipedia describes it[WPDNF]:

> In boolean logic, a disjunctive normal form (DNF) is a canonical normal form of a logical formula consisting of a disjunction of conjunctions; it can also be described as an OR of ANDs, a sum of products. [...]
>
> A logical formula is considered to be in DNF if it is a disjunction of one or more conjunctions of one or more literals. A DNF formula is in full disjunctive normal form if each of its variables appears exactly once in every conjunction and each conjunction appears at most once (up to the order of variables). As in conjunctive normal form (*CNF*), the only propositional operators in DNF are and ($\wedge$), or ($\vee$), and not ($\neg$). The not operator can only be used as part of a literal, which means that it can only precede a propositional variable.

DPLL algorithm This algorithm ist utterly irrelevant to the *satoku matrix*.

> As Wikipedia describes it[WPDPLL]:

In logic and computer science, the Davis–Putnam–Logemann–Loveland (DPLL) algorithm is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. for solving the CNF-SAT problem.

In logic and computer science, the Davis–Putnam–Logemann–Loveland (DPLL) algorithm is a complete, backtracking-based search algorithm for deciding the satisfiability of propositional logic formulae in conjunctive normal form, i.e. for solving the CNF-SAT problem.

The basic backtracking algorithm runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value. This is known as the splitting rule, as it splits the problem into two simpler sub-problems. The simplification step essentially removes all clauses that become true under the assignment from the formula, and all literals that become false from the remaining clauses.

The DPLL algorithm enhances over the backtracking algorithm by the eager use of the following rules at each step:

Unit propagation

If a clause is a unit clause, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true. Thus, no choice is necessary. Unit propagation consists in removing every clause containing a unit clause's literal and in discarding the complement of a unit clause's literal from every clause containing that complement. In practice, this often leads to deterministic cascades of units, thus avoiding a large part of the naive search space.

Pure literal elimination

If a propositional variable occurs with only one polarity in the formula, it is called pure. A pure literal can always be assigned in a way that makes all clauses containing it true. Thus, when it is assigned in such a way, these clauses do not constrain the search anymore, and can be deleted.

Unsatisfiability of a given partial assignment is detected if one clause becomes empty, i.e. if all its variables have been assigned in a way that makes the corresponding literals false. Satisfiability of the formula is detected either when all variables are assigned without generating the empty clause, or, in modern implementations, if all clauses are satisfied. Unsatisfiability of the complete formula can only be detected after exhaustive search.

Implicant As Wikipedia describes it[WPIMP]:

In Boolean logic, the term implicant has either a generic or a particular meaning. In the generic use, it refers to the hypothesis of an implication (implicant). In the particular use, a product term (i.e., a conjunction of literals) P is an implicant of a Boolean function F, denoted $P \leq F$, if P implies F (i.e., whenever P takes the value 1 so does F).

Implicate An implicate is a sum term (i.e., a disjunction of literals) P which implies falsehood of a Boolean function F. I.e., whenever P takes the value 0 so does F.

BIBLIOGRAPHY

[SHINY] Shiny, A. K., & Pujari, A. K. (1999). An Efficient Algorithm to Generate Prime Implicants. *Journal of Automated Reasoning*, *22*(2), 149–170. https://doi.org/10.1023/A:1005940031099.

In this paper, an efficient recursive algorithm is presented to compute the set of prime implicants of a propositional formula in conjunctive normal form (CNF). The propositional formula is represented as a (0,1)-matrix, and a set of 1's across its columns are termed as paths. The algorithm finds the prime implicants as the prime paths in the matrix using the divide-and-conquer technique. The algorithm is based on the principle that the prime implicant of a formula is the concatenation of the prime implicants of two of its subformulae. The set of prime paths containing a specific literal and devoid of a literal are characterized. Based on this characterization, the formula is recursively divided into subformulae to employ the divide-and-conquer paradigm. The prime paths of the subformulae are then concatenated to obtain the prime paths of the formula. In this process, the number of subsumption operations is reduced. It is also shown that the earlier algorithm based on prime paths has some avoidable computations that the proposed algorithm avoids. Besides being more efficient, the proposed algorithm has the additional advantage of being suitable for the incremental method, without recomputing prime paths for the updated formula. The subsumption operation is one of the crucial operations for any such algorithms, and it is shown that the number of subsumption operation is reduced in the proposed algorithm. Experimental results are presented to substantiate that the proposed algorithm is more efficient than the existing algorithms.

[WPADJ] Wikipedia contributors. (2025, March 28). Adjacency matrix. In Wikipedia, The Free Encyclopedia. Retrieved 20:27, April 11, 2025, from https://en.wikipedia.org/w/index.php?title=Adjacency_matrix&oldid=1282826257

[WPBCF] Wikipedia contributors. (2025, March 23). Blake canonical form. In Wikipedia, The Free Encyclopedia. Retrieved 15:29, April 10, 2025, from https://en.wikipedia.org/w/index.php?title=Blake_canonical_form&oldid=1281935421

[WPCNF] Wikipedia contributors. (2025, February 11). Conjunctive normal form. In Wikipedia, The Free Encyclopedia. Retrieved 20:27, April 10, 2025, from https://en.wikipedia.org/w/index.php?title=Conjunctive_normal_form&oldid=1275190406

[WPDNF] Wikipedia contributors. (2025, April 4). Disjunctive normal form. In Wikipedia, The Free Encyclopedia. Retrieved 01:46, April 11, 2025, from https://en.wikipedia.org/w/index.php?title=Disjunctive_normal_form&oldid=1283900821

[WPDPLL] Wikipedia contributors. (2025, February 21). DPLL algorithm. In Wikipedia, The Free Encyclopedia. Retrieved 04:30, April 12, 2025, from https://en.wikipedia.org/w/index.php?title=DPLL_algorithm&oldid=1276987229

[WPIMP] Wikipedia contributors. (2025, January 14). Implicant. In Wikipedia, The Free Encyclopedia. Retrieved 14:49, April 10, 2025, from https://en.wikipedia.org/w/index.php?title=Implicant&oldid=1269337610

[CHENG201115] Cheng, CK. (2011). Lecture 15: Karnaugh Maps II. In *CSE20, Discrete Mathematics*. University of California, San Diego. https://cseweb.ucsd.edu/classes/sp11/cse20-a/notes/lec15.ppt.

Objective of this course is to introduce discrete mathematics for computer system designs.

[deKleer1999] Kleer, J. de. (1999). An Improved Incremental Algorithm for Generating Prime Implicates. In H. J. Levesque & F. Pirri (Eds.), *Logical Foundations for Cognitive Agents: Contributions in Honor of Ray Reiter* (pp. 103–112). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-60211-5_9.

In 1987 Ray Reiter and I wrote a paper entitled "Foundations of assumption-based truth maintenance systems: Preliminary report" which showed how the behavior of the Assumption-Based Truth Maintenance System can be defined using the notions of prime implicate and prime implicant. This definition of the ATMS immediately suggests generalizing the ATMS to operate on arbitrary clauses. This generalization raises two immediate computational challenges (to me, not Ray who seems immune to such challenges). First, computing prime implicates/implicants is very expensive. Second, since ATMS's are used incrementally we need to exploit previous computation. This paper describes an improved and incremental algorithm to compute prime implicates/implicants. This algorithm allows us to experiment with the ideas Ray and I laid out in our paper. Unfortunately, the task is inherently NP-complete and all this paper can accomplish is present a more clever incremental algorithm.

[DELVAL1994551] del Val, A. (1994). Tractable Databases: How to Make Propositional Unit Resolution Complete through Compilation. In J. Doyle, E. Sandewall, & P. Torasso (Eds.), *Principles of Knowledge Representation and Reasoning* (pp. 551–561). Morgan Kaufmann. https://doi.org/10.1016/B978-1-4832-1452-8.50146-9.

We present procedures to compile any propositional clausal database σ into a logically equivalent "compiled" database σ* such that. for any clause C, σ |= C if and only if there is a unit refutation of σ* U C. It follows that once the compilation process is complete any query about the logical consequences of σ can be correctly answered in time linear in the sum of the sizes of σ* and the query. The compiled database σ* is for all but one of the procedures a subset of the set PI(σ) of prime implicates of σ, but σ* can be exponentially smaller than PI(σ). Of independent interest, we prove the equivalence of unit-refutability with two restrictions of resolution, and provide a new sufficient condition for unit refutation completeness, thus identifying a new class of tractable theories, one which is of interest to abduction problems as well. Finally, we apply the results to the design of a complete LTMS.

[Echenim2017PrimeIG] Echenim, M., Peltier, N., & Tourret, S. (2017). Prime Implicate Generation in Equational Logic. *J. Artif. Intell. Res.*, *60*, 827–880. https://api.semanticscholar.org/CorpusID:8314981.

We present an algorithm for the generation of prime implicates in equational logic, that is, of the most general consequences of formulæ containing equations and disequations between first-order terms. This algorithm is defined by a calculus that is proved to be correct and complete. We then focus on the case where the considered clause set is ground, i.e., contains no variables, and devise a specialized tree data structure that is designed to efficiently detect and delete redundant implicates. The corresponding algorithms are presented along with their termination and correctness proofs. Finally, an experimental evaluation of this prime implicate generation method is conducted in the ground case, including a comparison with state-of-the-art propositional and first-order prime implicate generation tools.

[KEAN1990185] Kean, A., & Tsiknis, G. (1990). An incremental method for generating prime implicants/implicates. *Journal of Symbolic Computation*, *9*(2), 185–206. https://doi.org/https://doi.org/10.1016/S0747-7171(08)80029-6.

Given the recent investigation of Clause Management Systems (CMSs) for ArtificialIntelligence applications, there is an urgent need for an efficient incremental method for generating prime implicants. Given a set of clauses F, a set of prime implicants II of F and a clause C1 the problem can be formulated as finding the set of prime implicants for II U C. Intuitively, the property of implicants, being prime implies that any effort to generate prime implicants from a set of prime implicants will not yield any new prime implicants but themselves. In this paper, we exploit the properties of prime implicants and propose an incremental method for generating prime implicants from a set of existing prime implicants plus a new clause. The correctness proof and complexity analysis of the incremental method are presented, and the intricacy of subsumptions in the incremental method is also examined. Additionally, the role of prime implicants in the CMS is also mentioned.

[MOSSE] Mossé, M., Sha, H., & Tan, L.-Y. (2022). A Generalization of the Satisfiability Coding Lemma and Its Applications. In K. S. Meel & O. Strichman (Eds.), *25th International Conference on Theory*

and Applications of Satisfiability Testing (SAT 2022) (Vol. 236, pp. 9:1–9:18). Schloss Dagstuhl – Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.SAT.2022.9.

The seminal Satisfiability Coding Lemma of Paturi, Pudlák, and Zane is a coding scheme for satisfying assignments of k-CNF formulas. We generalize it to give a coding scheme for implicants and use this generalized scheme to establish new structural and algorithmic properties of prime implicants of k-CNF formulas. Our first application is a near-optimal bound of $n \cdot 3^{n(1-\Omega(1/k))}$ on the number of prime implicants of any n-variable k-CNF formula. This resolves an open problem from the Ph.D. thesis of Talebanfard, who proved such a bound for the special case of constant-read k-CNF formulas. Our proof is algorithmic in nature, yielding an algorithm for computing the set of all prime implicants - the Blake Canonical Form BCF - of a given k-CNF formula. The problem of computing the Blake Canonical Form BCF of a given function is a classic one, dating back to Quine, and our work gives the first non-trivial algorithm for k-CNF formulas.

[MOUNT2012] David M. Mount. (2012). Lecture Notes. *CMSC 451, Design and Analysis of Computer Algorithms*. https://sw-amt.ws/David-Mount-cmsc451-lects.pdf original URL at https://cseweb.ucsd.edu/classes/sp11/cse20-a/notes/lec15.ppt is no longer available.

[SCH2013CDF] Scherer, W. (2013). *Generalization of CNF and Consequences for DNF of Implicants under Distributive Expansion*. https://sw-amt.ws/satoku/doc/doc-cnf-cdf-pde/satoku-cnf-cdf-pde.pdf [Online; accessed 2015-02-01].

The conjunctive normal form CNF, is generalized to a conjunction of disjunctive normal form clauses CDF, by dropping the restrictions for syllogistic formulas. It is shown, that this can lead to more desirable results for solving some satisfiability and counting problems. Distributive expansion of logical formulae is shown to have properties different from distributive expansion of arithmetic formulae and can be broken down into polynomial time and exponential time parts. The polynomial time portion can be used to develop systematic algorithms which can neither be provided by mathematical logic nor plain graph theory.

[SCHLENGA2020] Schlenga, A. (2020). SAT - CDCL. *Seminar - Automated Reasoning*. https://www21.in.tum.de/teaching/sar/SS20/1.pdf [Online; accessed 2025-04-14].

I present the CDCL algorithm and its implementation based on existing literature. This algorithm is used to solve SAT problems efficiently. First, the basics of SAT solving are introduced, then the CDCL algorithm's functionality is explained and a modern implementation is presented

INDEX