

# M 輪講 Factoring integers

252305012 伊藤 碧己

2023 年 5 月 8 日

**Factoring integers.** Next on our agenda is a family of SAT instances with quite a different flavor. Given an  $(m + n)$ -bit binary integer  $z = (z_{m+n} \dots z_2 z_1)_2$ , do there exist integers  $x = (x_m \dots x_1)_2$  and  $y = (y_n \dots y_1)_2$  such that  $z = x \times y$ ? For example, if  $m = 2$  and  $n = 3$ , we want to invert the binary multiplication

$$\begin{array}{rcl} & y_3 & y_2 & y_1 \\ \times & x_2 & x_1 \\ \hline & a_3 & a_2 & a_1 \\ b_3 & b_2 & b_1 \\ \hline c_3 & c_2 & c_1 \\ \hline z_5 & z_4 & z_3 & z_2 & z_1 \end{array} \quad \begin{aligned} (a_3 a_2 a_1)_2 &= (y_3 y_2 y_1)_2 \times x_1 & z_1 &= a_1 \\ (b_3 b_2 b_1)_2 &= (y_3 y_2 y_1)_2 \times x_2 & (c_1 z_2)_2 &= a_2 + b_1 \\ & & (c_2 z_3)_2 &= a_3 + b_2 + c_1 \\ & & (c_3 z_4)_2 &= b_3 + c_2 \\ & & z_5 &= c_3 \end{aligned} \quad (22)$$

when the  $z$  bits are given. This problem is satisfiable when  $z = 21 = (10101)_2$ , in the sense that suitable binary values  $x_1, x_2, y_1, y_2, y_3, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$  do satisfy these equations. But it's unsatisfiable when  $z = 19 = (10011)_2$ .

因数分解. 次のテーマは、まったく異なったテイストの SAT インスタンスファミリーである.  $(m + n)$ bit の 2 進整数  $z = (z_{m+n} \dots z_2 z_1)_2$  が与えられた時, そのような整数  $x = (x_m \dots x_1)_2$ ,  $y = (y_n \dots y_1)_2$  は存在するだろうか,  $z = x \times y$  となる. 例えば,  $m=2, n=3$  の時, 2 進数の乗算を転換させたい. <sup>\*1</sup>  $z$  ビット <sup>\*2</sup> が与えられた時. この問題は  $z = 21 = (10101)_2$  の時充足される, 適切な 2 進数  $x_1, x_2, y_1, y_2, y_3, a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3$  がこれらの等式<sup>\*3</sup>を満たすという意味で. <sup>\*4</sup> しかし,  $z = 19 = (10011)_2$  のときは充足不能である.

<sup>\*1</sup> 乗算の形から因数分解を考えるという意味?

<sup>\*2</sup> 即ち 2 進整数  $z$

<sup>\*3</sup> (22) の等式

<sup>\*4</sup>  $x = (11)_2, y = 7(111)_2$  すなわち  $a_1 = 1, a_2 = 1, a_3 = 1, b_1 = 1, b_2 = 1, b_3 = 1, c_1 = 1, c_2 = 1, c_3 = 1, z_1 = 1, z_2 = 0, z_3 = 1, z_4 = 0, z_5 = 1$

Arithmetical calculations like (22) are easily expressed in terms of clauses that can be fed to a SAT solver: We first specify the computation by constructing a Boolean chain, then we encode each step of the chain in terms of a few clauses. One such chain, if we identify  $a_1$  with  $z_1$  and  $c_3$  with  $z_5$ , is

$$\begin{aligned} z_1 &\leftarrow x_1 \wedge y_1, & b_1 &\leftarrow x_2 \wedge y_1, & z_2 &\leftarrow a_2 \oplus b_1, & s &\leftarrow a_3 \oplus b_2, & z_3 &\leftarrow s \oplus c_1, & z_4 &\leftarrow b_3 \oplus c_2, \\ a_2 &\leftarrow x_1 \wedge y_2, & b_2 &\leftarrow x_2 \wedge y_2, & c_1 &\leftarrow a_2 \wedge b_1, & p &\leftarrow a_3 \wedge b_2, & q &\leftarrow s \wedge c_1, & z_5 &\leftarrow b_3 \wedge c_2, \\ a_3 &\leftarrow x_1 \wedge y_3, & b_3 &\leftarrow x_2 \wedge y_3, & & & & & c_2 &\leftarrow p \vee q, \end{aligned} \quad (23)$$

using a “full adder” to compute  $c_2z_3$  and “half adders” to compute  $c_1z_2$  and  $c_3z_4$  (see 7.1.2-(23) and (24)). And that chain is equivalent to the 49 clauses

$$(x_1 \vee \bar{z}_1) \wedge (y_1 \vee \bar{z}_1) \wedge (\bar{x}_1 \vee \bar{y}_1 \vee z_1) \wedge \dots \wedge (\bar{b}_3 \vee \bar{c}_2 \vee \bar{z}_4) \wedge (b_3 \vee \bar{z}_5) \wedge (c_2 \vee \bar{z}_5) \wedge (\bar{b}_3 \vee \bar{c}_2 \vee z_5)$$

obtained by expanding the elementary computations according to simple rules:

$$\begin{aligned} t \leftarrow u \wedge v &\text{ becomes } (u \vee \bar{t}) \wedge (v \vee \bar{t}) \wedge (\bar{u} \vee \bar{v} \vee t); \\ t \leftarrow u \vee v &\text{ becomes } (\bar{u} \vee t) \wedge (\bar{v} \vee t) \wedge (u \vee v \vee \bar{t}); \\ t \leftarrow u \oplus v &\text{ becomes } (\bar{u} \vee v \vee t) \wedge (u \vee \bar{v} \vee t) \wedge (u \vee v \vee \bar{t}) \wedge (\bar{u} \vee \bar{v} \vee \bar{t}). \end{aligned} \quad (24)$$

(22) のような算術的な計算は、SAT ソルバーに与えることができる節として簡単に表現することができる: まず、ブール鎖<sup>\*5</sup>を構築することによってその計算を明記し、その鎖の各ステップをいくつかの節で符号化する。そのような鎖の1つは、 $a_1$ を $z_1$ 、 $c_3$ を $z_5$ とすると、 $c_2z_3$ の計算に「全加算器」を、 $c_1z_2$ と $c_3z_4$ の計算に「半加算器」を使用している(7.1.2-(23), (24)を参照)。そしてその鎖は49の節 $(x_1 \vee \bar{z}_1) \wedge (y_1 \vee \bar{z}_1) \wedge \dots \wedge (\bar{b}_3 \vee \bar{c}_2 \vee \bar{z}_5)$ と同値である、簡単なルール(24)で基礎的な計算を拡張することによって得られる。<sup>\*6 \*7</sup>

<sup>\*8</sup>

<sup>\*5</sup> ブール代数において、複数の真偽値を論理演算子でつないで一つの式として表したもの

<sup>\*6</sup> (24)のルールは、「 $\leftarrow$ 」を同値の記号「 $\leftrightarrow$ 」と置き換えて考えれば理解しやすい

<sup>\*7</sup>  $\oplus$ ...XOR(排他的論理和)

<sup>\*8</sup>  $A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$

To complete the specification of this factoring problem when, say,  $z = (10101)_2$ , we simply append the unary clauses  $(z_5) \wedge (\bar{z}_4) \wedge (z_3) \wedge (\bar{z}_2) \wedge (z_1)$ .

Logicians have known for a long time that computational steps can readily be expressed as conjunctions of clauses. Rules such as (24) are now called *Tseytin encoding*, after Gregory Tseytin (1966). Our representation of a small five-bit factorization problem in 49+5 clauses may not seem very efficient; but we will see shortly that  $m$ -bit by  $n$ -bit factorization corresponds to a satisfiability problem with fewer than  $6mn$  variables, and fewer than  $20mn$  clauses of length 3 or less.

*Even if the system has hundreds or thousands of formulas,  
it can be put into the conjunctive normal form "piece by piece",  
without any "multiplying out."*

— MARTIN DAVIS and HILARY PUTNAM (1958)

この因数分解問題の仕様を完成させるために、例えば、 $z = (10101)_2$  のとき、単に単位節  $z_5 \wedge \bar{z}_4 \wedge z_3 \wedge \bar{z}_2 \wedge z_1$  を追加する。

論理学者たちは、長い間知っていた、計算ステップが節の連結として容易に表現できることを。(24)のようなルールは現在、Tseytin 変換と呼ばれている、Gregory Tseytin(1966)<sup>\*9</sup>にちなんで。5ビットの小さな因数分解問題を  $49 + 5$  の節で表現したのは、あまり効率的ではないかもしれない；しかし、まもなく分かる、 $m$ ビット  $\times n$ ビットの因数分解は SAT 問題に対応する、 $6mn$ 以下の変数と  $20mn$ 以下の長さ 3以下の節の。—たとえ数百、数千の式を持つシステムでも、”1つ1つ”連言標準形に落とし込むことができる、拡大することなく—

— MARTIN DAVIS<sup>\*10</sup> and HILARY PUTNAM<sup>\*11</sup> (1958)

<sup>\*9</sup> Gregory Tseytin(1936-2022): ロシアの数学者およびコンピューター科学者.Tseytin 変換の生みの親

<sup>\*10</sup> Martin Davis(1928-2023): アメリカの数学者およびコンピューター科学者.DPLL アルゴリズムの共同開発者

<sup>\*11</sup> Hilary Putnam(1926-2016): スコットランドの哲学者およびコンピューター科学者.DPLL アルゴリズムの共同開発者

Suppose  $m \leq n$ . The easiest way to set up Boolean chains for multiplication is probably to use a scheme that goes back to John Napier's *Rabdologiæ* (Edinburgh, 1617), pages 137–143, as modernized by Luigi Dadda [*Alta Frequenza* **34** (1964), 349–356]: First we form all  $mn$  products  $x_i \wedge y_j$ , putting every such bit into  $\text{bin}[i + j]$ , which is one of  $m + n$  “bins” that hold bits to be added for a particular power of 2 in the binary number system. The bins will contain respectively  $(0, 1, 2, \dots, m, m, \dots, m, \dots, 2, 1)$  bits at this point, with  $n-m+1$  occurrences of “ $m$ ” in the middle. Now we look at  $\text{bin}[k]$  for  $k = 2, 3, \dots$ . If  $\text{bin}[k]$  contains a single bit  $b$ , we simply set  $z_{k-1} \leftarrow b$ . If it contains two bits  $\{b, b'\}$ , we use a half adder to compute  $z_{k-1} \leftarrow b \oplus b'$ ,  $c \leftarrow b \wedge b'$ , and we put the carry bit  $c$  into  $\text{bin}[k+1]$ . Otherwise  $\text{bin}[k]$  contains  $t \geq 3$  bits; we choose any three of them, say  $\{b, b', b''\}$ , and remove them from the bin. With a full adder we then compute  $r \leftarrow b \oplus b' \oplus b''$  and  $c \leftarrow \langle bb'b'' \rangle$ , so that  $b + b' + b'' = r + 2c$ ; and we put  $r$  into  $\text{bin}[k]$ ,  $c$  into  $\text{bin}[k+1]$ . This decreases  $t$  by 2, so eventually we will have computed  $z_{k-1}$ . Exercise 41 quantifies the exact amount of calculation involved.

$m \leq n$  とする。掛け算のブール鎖を用意する最も簡単な方法はおそらく、John Napier<sup>\*12</sup>の *Rabdologiæ* (Edin-burgh, 1617)<sup>\*13</sup>に遡る方式を使うことだろう、その 137-143 ページを Luigi Dadda によって現代風にアレンジされたもの [*Alta Frequenza* 34 (1964), 349-356] として：まず、 $mn$  個の積  $x_i \wedge y_j$  をすべて形成し、そのような全てのビットを  $\text{bin}[i + j]$  に格納する。これは  $m + n$  個の「BIN」の 1 つである、2 進数システムの特定の 2 のべき乗に対して追加されるビットを保持する。BIN は、この時点ではそれ (0, 1, 2, ..., m, m, ..., 2, 1) ビットを含み、真ん中では  $n-m+1$  回の” $m$ ”が出現する。ここで、 $k = 2, 3, \dots$  の場合の  $\text{bin}[k]$  を確認する。もし  $\text{bin}[k]$  に 1 つのビット  $b$  が含まれている場合、単に  $z_{k-1} \leftarrow b$  をセットし、2 つのビット  $\{b, b'\}$  が含まれている場合は、半加算器で  $z_{k-1} \leftarrow b \oplus b'$ ,  $c \leftarrow b \wedge b'$  を計算し、キャリービット  $c$  を  $\text{bin}[k+1]$  に入れる。それ以外の場合、 $\text{bin}[k]$  には  $t \geq 3$  ビットが含まれる。そのうちの任意の 3 つ、例えば  $\{b, b', b''\}$  を選び、BIN から削除する。そして全加算器によって  $r \leftarrow b \oplus b' \oplus b''$  と  $c \leftarrow \langle bb'b'' \rangle$ <sup>\*14</sup>を計算すると、 $b + b' + b'' = r + 2c$  となるので<sup>\*15</sup>、 $r$  を  $\text{bin}[k]$  に、 $c$  を  $\text{bin}[k+1]$  に入れる。これで  $t$  が 2 減るので、最終的には  $z_{k-1}$  を計算したことになる。練習問題 41 は、正確な計算量を数値化したものある。

\*12 John Napier(1550-1617): スコットランドの数学者。対数を考案した

\*13 1617 年エディンバラで出版された John Napier のラブドロジーという論文。算術計算を支援する 3 つの装置について説明している

\*14  $\langle \rangle$  の意味は調べてもわからなかった。計算の意味を考えると  $\langle \rangle$  の中のブール変数のうち 2 つ以上が真なら真と考察できる

\*15  $r+2c$  の  $2c$  というのは、2 進数で 1 つ上の桁に  $c$  を足すことを意味している (memo)

37.5'  $m=3, n=4$

①  $m \times n$  個の積  $x_i \wedge y_j$  を形成'. ( $1 \leq i \leq m, 1 \leq j \leq n$ )

$$x_1 \wedge y_1, x_1 \wedge y_2, x_1 \wedge y_3, x_1 \wedge y_4$$

$$x_2 \wedge y_1, x_2 \wedge y_2, x_2 \wedge y_3, x_2 \wedge y_4$$

$$x_3 \wedge y_1, x_3 \wedge y_2, x_3 \wedge y_3, x_3 \wedge y_4$$

②, ① の積を,  $\text{bin}[i+j]$  に格納.

$$\text{bin}[1] = \text{none}$$

$$\text{bin}[6] = x_2 \wedge y_4, x_3 \wedge y_3$$

$$\text{bin}[2] = x_1 \wedge y_1$$

$$\text{bin}[7] = x_3 \wedge y_4$$

$$\text{bin}[3] = x_1 \wedge y_2, x_2 \wedge y_1$$

$$\text{bin}[4] = x_1 \wedge y_3, x_2 \wedge y_2, x_3 \wedge y_1$$

$$\text{bin}[5] = x_1 \wedge y_4, x_2 \wedge y_3, x_3 \wedge y_2$$

③  $\text{bin}[k]$  ( $k=2, 3, \dots$ ) (= ついつ. ( $\text{bin}[k]$  の中身を  $b, b', b'' \dots$  とする))

3.1.  $|\text{bin}[k]| = 1$  の場合,  $\exists k-1 \leftarrow b$

3.2.  $|\text{bin}[k]| = 2$  の場合,  $\exists k-1 \leftarrow b \oplus b', c \leftarrow b \wedge b'$

3.3.  $|\text{bin}[k]| \geq 3$  の場合, 任意の  $b, b', b''$  (= ついて)

$$\exists r \leftarrow b \oplus b' \oplus b'', c \leftarrow \langle b b' b'' \rangle \\ (= b \wedge b' \wedge b'')$$

を行.,  $b, b', b'' \in \text{bin}[k]$  が消去

3.1 ~ 3.3 以後,  $r \in \text{bin}[k]$  は,  $c \in \text{bin}[k+1]$  は入る,

3.3 を行. た. 場合は ③ を繰り返す。

図 1 計算の例 ( $m=3, n=4$ )

$$x = (x_3 \ x_2 \ x_1)_2$$

$$y = (y_4 \ y_3 \ y_2 \ y_1)_2 \text{ ので, } a_{ij} = x_i \wedge y_j \text{ とする},$$

$$\begin{array}{r} x \times y \\ \begin{array}{cccc} y_4 & y_3 & y_2 & y_1 \\ \times & x_3 & x_2 & x_1 \\ \hline a_{14} & a_{13} & a_{12} & a_{11} \\ a_{24} & a_{23} & a_{22} & a_{21} \\ \hline a_{34} & a_{33} & a_{32} & a_{31} \\ \hline z_6 & z_5 & z_4 & z_3 & z_2 & z_1 \end{array} \end{array}$$

$$z_1 \leftarrow a_{11}$$

$$z_2 \leftarrow a_{12} \oplus a_{21}$$

$$c_2 \leftarrow a_{12} \wedge a_{21}$$

$$r_3 \leftarrow a_{13} \oplus a_{22} \oplus a_{31}$$

$$c_{31} \leftarrow \langle a_{13} \ a_{22} \ a_{31} \rangle \text{ (と書いた...?)}$$

$$= (a_{13} \wedge a_{22}) \vee (a_{13} \wedge a_{31}) \vee (a_{22} \wedge a_{31})$$

$$z_3 \leftarrow r_2 \oplus c_2 \wedge a_{22}$$

$$c_{32} \leftarrow r_2 \wedge c_2$$

⋮

図2 計算例: ステップ3

This method of encoding multiplication into clauses is quite flexible, since we're allowed to choose *any* three bits from  $\text{bin}[k]$  whenever four or more bits are present. We could use a first-in-first-out strategy, always selecting bits from the "rear" and placing their sum at the "front"; or we could work last-in-first-out, essentially treating  $\text{bin}[k]$  as a stack instead of a queue. We could also select the bits randomly, to see if this makes our SAT solver any happier. Later in this section we'll refer to the clauses that represent the factoring problem by calling them  $\text{factor\_fifo}(m, n, z)$ ,  $\text{factor\_lifo}(m, n, z)$ , or  $\text{factor\_rand}(m, n, z, s)$ , respectively, where  $s$  is a seed for the random number generator used to generate them.

乗算を節に符号化するこの方法は非常に柔軟である。我々は  $\text{bin}[k]$  からどの 3 ビットでも選択することができるから。4 つや、それ以上のビットが存在するときでも。我々は先入れ先出しの戦略を使うことができた、常に”後ろ”からビットを選択し、その和を”前”に配置する；あるいは、後入れ先出しで動作させ、実質的に  $\text{bin}[k]$  をキューではなくスタックとして扱うこともできた。<sup>\*16</sup>また、ビットをランダムに選択し、それが我々の SAT ソルバーをより幸せにするかどうかを確認することもできる。このセクションの後半では、因数分解問題を表す節について参照する、それぞれ  $\text{factor\_fifo}(m, n, z)$ ,  $\text{factor\_lifo}(m, n, z)$ ,  $\text{factor\_rand}(m, n, z, s)$  と呼び、 $s$  は生成に使用する乱数ジェネレーターのシードであるとする。

---

<sup>\*16</sup> キューは最初に入れたデータを最初に取り出し (FIFO), スタックは最後に入れたデータを最初に取り出す (LIFO) データの取り出し手順

It's somewhat mind-boggling to realize that numbers can be factored without using any number theory! No greatest common divisors, no applications of Fermat's theorems, etc., are anywhere in sight. We're providing no hints to the solver except for a bunch of Boolean formulas that operate almost blindly at the bit level. Yet factors are found.

数字を何の数論も用いずに因数分解できることに、幾分驚かされる！最大公約数もなければ、フェルマーの定理の応用もなく、他にも何もない…、それらはどこにでもあるようなものである。我々は SAT ソルバーへのヒントは全く与えていない布尔式の束を除いて、ビットレベルでほとんど盲目的に動作する。それでも因数は見つかる。

Of course we can't expect this method to compete with the sophisticated factorization algorithms of Section 4.5.4. But the problem of factoring does demonstrate the great versatility of clauses. And its clauses can be combined with other constraints that go well beyond any of the problems we've studied before.

もちろん、我々は予期できない、この方法が 4.5.4 節の高度な因数分解アルゴリズムと競合することを。しかし、因数分解の問題は、節の素晴らしい汎用性を示している。そしてそれらの節は他の制約と組み合わせることができ、我々がこれまで研究してきたどの問題よりもはるかに優れている。

**41.** [M21] Determine the number of Boolean operations  $\wedge$ ,  $\vee$ ,  $\oplus$  needed to multiply  $m$ -bit numbers by  $n$ -bit numbers with Dadda's scheme, when  $2 \leq m \leq n$ .

Ex41. ブール演算子  $\wedge$ ,  $\vee$ ,  $\oplus$  の数を決定せよ. Dadda の方式で  $m$  ビットの数を  $n$  ビット数で乗算するのに必要な,  $2 \leq m \leq n$  のとき.

**41.** First there are  $mn$  ANDs to form  $x_i y_j$ . A bin that contains  $t$  bits initially will generate  $\lfloor t/2 \rfloor$  carries for the next bin, using  $(t-1)/2$  adders. (For example,  $t=6$  will invoke 2 full adders and one half adder.) The respective values of  $t$  for  $\text{bin}[2]$ ,  $\text{bin}[3]$ , ...,  $\text{bin}[m+n+1]$  are  $(1, 2, 4, 6, \dots, 2m-2, 2m-1, \dots, 2m-1, 2m-2, 2m-3, \dots, 5, 3, 1)$ , with  $n-m$  occurrences of  $2m-1$ . That makes a total of  $mn - m - n$  full adders and  $m$  half adders; altogether we get  $mn + 2(mn - m - n) + m$  instances of AND,  $mn - m - n$  instances of OR, and  $2(mn - m - n) + m$  instances of XOR.

まず、 $x_i y_j$  を形成するために  $mn$  個の AND がある.  $t$  個のビットを含む bin は最初に, 次の bin のために  $\lfloor t/2 \rfloor$  個のキャリーを生成する,  $(t-1)/2$  個の加算器を使って.\*<sup>17</sup>(例えば,  $t = 6$  の場合 2 つの全加算器と 1 つの半加算器を呼び出す)\*<sup>18</sup> \*<sup>19</sup>  $\text{bin}[2]$ ,  $\text{bin}[3]$ , ...,  $\text{bin}[m+n+1]$  のそれぞれの  $t$  の値は,  $(1, 2, 4, 6, \dots, 2m-2, 2m-1, \dots, 2m-1, 2m-2, 2m-3, \dots, 5, 3, 1)$  となり,  $2m-1$  は  $n-m$  回発生する. それは合計  $mn-m-n$  の全加算器と  $m$  の半加算器を作り, 併せて  $mn + 2(mn - m - n) + m$  個の AND のインスタンスと,  $mn - m - n$  の OR のインスタンスと,  $2(mn - m - n) + m$  の XOR のインスタンスを得られる.

---

\*<sup>17</sup> ここで言う adders は全加算器のこと

\*<sup>18</sup> 全加算器は下からの繰り上がりを考慮し, 3 つを加算するもの, 半加算器は 2 つを加算するもの

\*<sup>19</sup> 即ち  $t=6$  の場合, (i)3 つ選んで全加算器で計算 ( $t$  が 4 になる) (ii)3 つ選んで全加算器で計算 ( $t$  が 2 になる) (iii) 残り 2 つを半加算器で計算となる

Ex 4)

$$\textcircled{1} \quad x = (x_m x_{m-1} \dots x_2 x_1)_2$$

$y = (y_n y_{n-1} \dots y_2 y_1)_2$  とする。

$x_i y_j$  ( $x_i \wedge y_j$ ) を作る。 $(1 \leq i \leq m, 1 \leq j \leq n)$

これは、 $m n$  個の AND が存在する。

\textcircled{2}  $x_i y_j \in \text{bin}[i+j] = 1$  である。

\textcircled{3}  $\text{bin}[k]$  ( $k = 2, 3, \dots, m+n+1$ ) について。

(i)  $|\text{bin}[k]| \geq 3$  のとき、 $\text{bin}[k]$  の任意の 3 つの要素を全加算器で足し、 $\text{bin}[k+1]$  にキャリービットを、 $\text{bin}[k]$  に出力ビットを足す。

(ii)  $|\text{bin}[k]| = 2$  のとき、 $\text{bin}[k]$  の 2 つの要素を半加算器で足し、 $\text{bin}[k+1]$  にキャリービットを、 $\text{bin}[k]$  に出力ビットを足して終了。

(iii)  $|\text{bin}[k]| = 1$  のとき、 $\text{bin}[k] \leftarrow b$  ( $b$  は  $\text{bin}[k]$  の要素) とし終了。

終了するまで (i) ~ (iii) の操作をくりかえす。

ここで、7 個のビットを含む  $\text{bin}$  は、

(i)  $t = 2N$  ( $N$  は自然数) のとき、

$N-1$  回の (i) の操作を行い、(ii) の操作を行って終了するので、

生成するキャリービットの数は  $N$  ( $= \lfloor t/2 \rfloor$ )

(ii)  $t = 2N-1$  のとき、

$N-1$  回の (i) の操作を行い、(iii) の操作を行って終了するので、

生成するキャリービットの数は  $N-1$  ( $= \lfloor t/2 \rfloor$ )

よって、7 個のビットを含む  $\text{bin}$  が生成するキャリービットの数は

$$\lfloor t/2 \rfloor$$

Date

$n=6, m=4$  の場合

0	0	0	0	0	0
0	0	0	0		
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0

(2 3 4 4 4 3 2 1) ←  $\pi_2, \text{bin}[k]$  の数  
k=n+3~ k=2~m+1 bin  
m+n k=m+2~  
n+2

キャリービットを足し  $\pi_2$  後の  $\text{bin}[k]$  の数を  $\text{bin}'[k]$  とする。

$$\text{bin}'[k] = \text{bin}[k] + \lfloor \text{bin}[k]/2 \rfloor \text{ とする。}$$

$$\text{bin}[k] = (1, 2, 3, 4, 4, 4, 3, 2, 1) \text{ とする。}$$

$$\text{bin}'[k] = (1, 2, 4, 6, 7, 7, 6, 5, 3, 1)$$

これを一般化する。

$$\text{bin}[k] = (1, 2, 3, \dots, m, m, m, m, m-1, \dots, 1)$$

$m \approx$        $m-m$        $m$        $\approx m-2, \dots$

$$\text{bin}'[k] = (1, 2, 4, 6, \dots, 2m-2, 2m-1, 2m-1, 2m-1, 2m-2, 2m-3, 5, 3, 1)$$

$m \approx$        $\approx m-3,$        $k=n+4 \sim m+1$

ここで、 $\text{bin}[k]$ が計算に用いる全加算器数は、1回の(i)の操作ごとに  $t$  が 2 減るため、

・  $t$  が奇数のとき、 $t - 1/2$

・  $t$  が偶数のとき、 $t/2 - 1$

また、 $t$  が偶数のとき、 $t/2 - 1$  回の(i)の操作後(ii)の操作を行なう。半加算器を 1つ用いる。

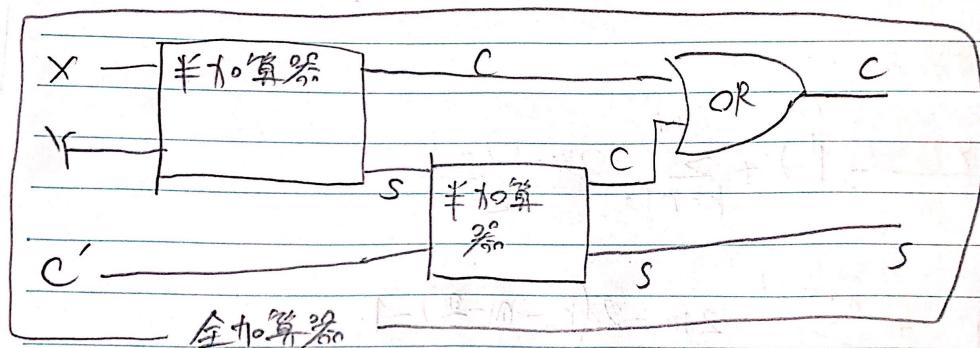
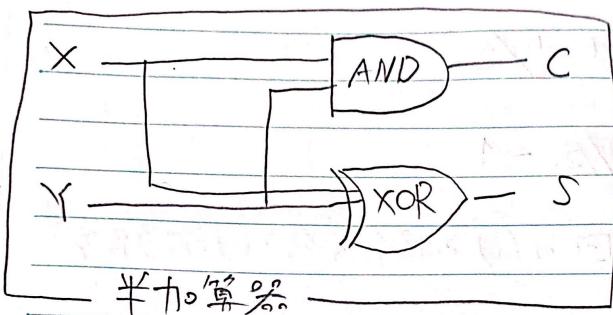
よって、用いる全加算器の合計は、

$$\begin{aligned}
 & \sum_{k=3}^{m+1} \left( \frac{2k-4}{2} - 1 \right) + \sum_{k=m+2}^{n+1} \frac{(2m-1)-1}{2} \\
 & + \frac{2m-2}{2} - 1 + \sum_{k=h+3}^{h+m+1} \frac{2m-2(k-n-2)-1}{2} \\
 = & \left\{ \frac{1}{2}(m+1)(m+2) - 3(m+1) \right\} + 3 + (m-1)(n-m) \\
 & + m-2 + \frac{1}{2}m(m-1) - (m-1) \\
 = & \frac{1}{2}m^2 + \frac{3}{2}m + 1 - 3m - 3 + 3 + nm - m^2 - n + m \\
 = & nm - n - m
 \end{aligned}$$

No. \_\_\_\_\_

Date \_\_\_\_\_

また、七ヶ島偶数の bin[K] は  $m$  個あるので、  
用いる半加算器の数は  $m$



全加算器は 2つ半加算器と 1つ ORゲート、  
半加算器は 1つ ANDゲートと 1つ XORゲートを用いる。

用いる OR の 数は  $nm - h - m$

合計で用いる半加算器の数は  $2nm - 2n - m$

用いる AND の 数は  $2nm - 2n - m + nm = nm - 2n - m$

用いる XOR の 数は  $2nm - 2n - m$

```
[base] ramurezun@ITO-MacBook-Pro Factoring integers % python factoring.py 18 2 3
num_half_adder: 4
num_full_adder: 1
num_and: 10
num_or: 1
num_xor: 4
公式から求めたhalf_adder: 4
公式から求めたnum_full_adder: 1
公式から求めたand: 10
公式から求めたor: 1
公式から求めたxor: 4
[base] ramurezun@ITO-MacBook-Pro Factoring integers % python make_csp.py 18 2 3 >test.csp
[base] ramurezun@ITO-MacBook-Pro Factoring integers % sugar test.csp
s SATISFIABLE
a x_1    true
a x_2    true
a y_1    false
a y_2    true
a y_3    true
a
```

図3 論

理記号の数を求めるプログラムと CSP 形式で出力するプログラム