

# Transceternal Codegolf Technique

December 26, 2020

# 概要

---

Transceternal でソースコードを短くしたいときの方針を紹介する。ライセンスは CC0-1.0<sup>1</sup> とする。

なお、ソースコードとは以下のような Transceternal 処理系に直接読み込むものを指す

```
catacat
```

実際に試す際にはソースコードの末尾に改行を含まないように注意。Transceternal 言語の基本的な仕様を把握していることを前提とする。言語原作者による解説

<https://esolangs.org/wiki/Transceternal> もしくは筆者のスライド <https://hiromi-mi.github.io/trans.pdf> を参照。

---

<sup>1</sup><http://creativecommons.org/publicdomain/zero/1.0/>

# Before 1560 バイト

---

0 1 2 2 2 3 4 5 3 6 3 7 3 8 3 9 3 A 3 B 3 C 3 D 3 E 3 F 3  
G 3 H 3 I 3 J 3 K 3 L 3 M 3 N 3 O 3 P 3 Q 3 R 3 S 3 T 3 U  
3 V 3 W 3 X 3 Y 3 Z 3 a 3 b 3 c 2 c d 3 2 d 2d 2e 2f 2f 2f  
2g 2h 1p 3 1q 3 1r 3 1s 3 1t 3 1u 3 1v 3 1w 3 1x 3 1y 2 1y  
1h 3 1i 3 1j 3 1k 3 1l 3 1m 3 1n 3 1o 3 1p 2Y 2Z 2a 2a 2a  
2b 2c 1o 1g 3 1h 2T 2U 2V 2V 2V 2W 2X 1n 1f 3 1g 20 2P 2Q  
2Q 2Q 2R 2S 1m 1e 3 1f 2J 2K 2L 2L 2L 2M 2N 1l 1d 3 1e 2E  
2F 2G 2G 2G 2H 2I 1k 1c 3 1d 29 2A 2B 2B 2B 2C 2D 1j 1b 3  
1c 24 25 26 26 26 27 28 1c 1k 1z 20 21 21 21 22 23 1a 3 1b  
1i e u 2 t f 2 g 2 h 3 i 3 j 3 j k 3 l 3 m 3 n 3 o 3 p 3 q  
3 r 3 s 3 s 2i 3o 2 3n 2j 3 2j 2k 3 2l 3 2m 3 2n 3 2o 3 2p  
3 2q 3 2r 3 2s 3 2t 3 2u 3 2v 3 2w 3 2x 3 2y 3 2z 3 30 3 31  
3 32 3 33 3 34 3 35 3 36 3 37 3 38 3 39 3 3A 3 3B 3 3C 3 3D  
3 3E 3 3F 3 3G 3 3H 3 3I 3 3J 3 3K 3 3L 3 3M 3 3N 3 3O 3 3P  
3 3Q 3 3R 3 3S 3 3T 3 3U 3 3V 3 3W 3 3X 3 3Y 3 3Z 3 3a 3 3b  
3 3c 3 3d 3 3e 3 3f 3 3g 3 3h 3 3i 3 3j 3 3k 3 3l 3 3m 3 3m  
3p 47 2 46 3q 2 3r 2 3s 3 3t 3 3t 3u 2 3v 2 3w 3 3x 3 3y 3  
3z 3 40 3 41 3 42 3 43 3 44 3 45 3 45 48 4N 2 4M 49 2 4A 2  
4B 3 4C 3 4D 3 4D 4E 2 4F 2 4G 2 4H 3 4I 2 4J 2 4K 2 4L 2 4L  
40 4d 2 4c 4P 2 4Q 2 4R 3 4S 2 4T 2 4U 3 4U 4V 2 4W 2 4X 3  
4Y 2 4Z 2 4a 3 4b 3 4b 4y 4z 50 50 50 51 52 4o 2 4p 2 4q 2  
4q 4r 2 4s 2 4t 3 4u 2 4v 2 4w 3 4x 2 4x 4e 4n 2 4m 4f 3 4f  
4g 2 4h 2 4i 3 4j 2 4k 3 4l 3 4l 2 2d v 1Z 2 1Y w 2 x 2 y 3  
z 3 10 3 10 11 3 12 3 13 3 14 3 15 3 16 3 17 3 18 3 19 3 1A  
3 1B 3 1C 3 1D 3 1E 3 1F 3 1G 3 1H 3 1I 3 1J 3 1K 3 1L 3 1M  
3 1N 3 1O 3 1P 3 1Q 3 1R 3 1S 3 1T 3 1U 3 1V 3 1W 3 1X 3 1X  
2i v v v v v v v v v

## After 231 バイト

---

012333435363738393a3b3c3d3e3f3g3h3i3j3k3l  
3m3n3o3p3q3r3s3 Φ 3u3v3w3x3y3z323IK2Jzr+,0-  
.W3X2X03P3Q3R3S3T3U3V3W'(0)\*VN30#\$0%&UM3N  
YZ0!"TL3MAE2DB2C2yz/=2<:2;2zs[^2]z\34\_~`Σ  
2~B{2|2}2}Y Ψ 2X t2:T 2B é ħ 0 í ũ {t Ω α  
2 Ÿ z İ 2t2+FH2GBb>@2?:4[FFF

(途中に空白文字があるように見られるのはフォントと  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  の設定の都合)

# 空白文字をなくす

---

**効果** 中規模以上のプログラムではソースコードを 2/3 程度に減らせる

**実装の手間** 普通

**適用可能性** いつでも

Transceternal の単一トークンは 1 文字 もしくは (スペース区切りの) 文字列 である

スペース区切りだとトークン文字数以外に空白も必要でバイト数がかさむ

Transceternal ではトークンの名前それ自体ではなく、トークン (つまり接点) 同士の関係にしか意味をもっていない

そこで、全てのトークンを 1 文字で表記するとスペース区切りが不要で、ソースコードの空白が縮むと考えられる

トークンの文字列としては印字可能マルチバイト文字も利用可能なのでロシア文字などの 2 バイト文字を使うと表記できる

## 空白文字をなくす

---

具体的には、なるべく多くの1バイト文字 (アルファベットと記号と数字全て) を使いつつ、使い切った後に2バイト文字を使うとよい Python で2バイト文字列を列挙するには (トークン数に応じ境界は調整すること):

```
# Cyriic, Arabic
"".join([chr(x) for x in range(0x3a3, 0x052f)]) + \
    "".join([chr(x) for x in range(0x61e, 0x70d)])
```

とできる

# 容量削減の方針

---

空白文字をなくした後、ソースコードのバイト数は

$$\text{節点数} \times \text{その節点のバイト数} \times 2$$

で決まる。

ソースコードを小さくするには、節点数を減らすこと、各節点のバイト数を減らすことが基本的な方針

アルゴリズムの改良もソースコードを小さくするには有用だが、ここでは取り上げず、Transceternal 固有の事項に絞り説明する  
中規模以上のプログラムでは、絶対アドレスの値を指定するための接点がボトルネックになる

# 絶対アドレス指定用接点の共有

---

効果 高い。半減も可能

実装の手間 特殊な場合の実装は簡単、一般の場合は手間？

適用可能性 いつでも

各コマンドでは左辺右辺として、deserialize したときにグラフ上の絶対アドレスを表す値を指定する。

指定するときのアドレスを表す値を、途中まで共有するようにすれば共有する分だけ節点数が削減できる

例: 10000, 010000 この2つのアドレスを表す値は、素直に表現すれば  $5 + 6 = 11$  節点必要だが、010000 を1のところで10000に繋げば  $5 + (6 - 5) = 6$  節点で済む。



## 絶対アドレス指定用接点の共有

---

多数の後方一致した絶対アドレス指定をしているほど効果が大きいが、一方アドレスが前方一致している場合は共有できないことに注意。

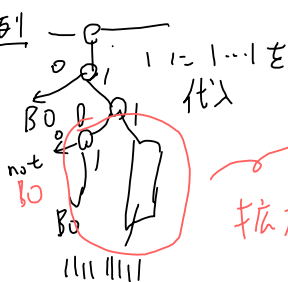
例: 1000, 1001 この2つのアドレスを表す値は、最後の 10 と 01 が異なる値をもっているので、共有できず  $4 + 4 = 8$  バイト必要。生成器で共有機構を完全に実装しようとするときは:

1. アドレスの逆順の suffix で二分木を構成し、二分木に対応する接点の位置を記録する。
2. アドレスを表す値を構築するときには、二分木を確認し二分木に後方部分文字列が一致すれば一致した二分木上の接点から 1...1 を構築する。構築後二分木に追記する

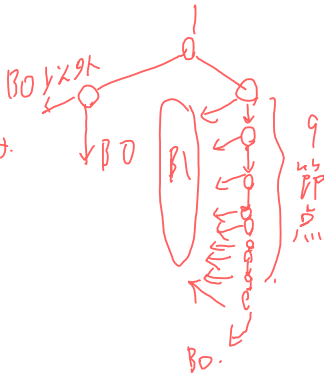
部分的に実装する場合 ( 1...1 という形のアドレスのみなど) は簡単で、1 が続くような値を生成するときに、生成した値とその個数を覚えておき、次回以降は記録したものに繋がればよい

## 絶対アドレス指定用接点の共有: 図解

例



扩大:



代入のための  
絶対アドレス指定  
に 10 節点使用

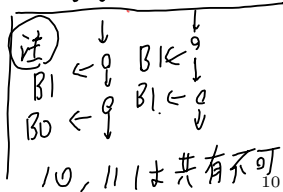
一方, deserialize  
時に「2」と認識され  
るためには別の  
節点である必要は  
ない

Before

After

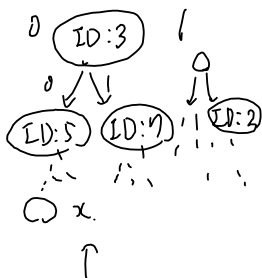


最後の「1」を共有するのは  
9 節点に減らせる!



# 絶対アドレス指定用接点の共有: 図解

## 表楚方法



Transceternal  
グラフ上の節点を  
IDとする

二分木Tをつくる。条件: 記された節点X  
から Transceternal Deserializateして「Tの葉から  
見た節点Xの位置」に一致する。ようにつくる。

新たな絶対アドレス指定をつくる時は、  
二分木をたどり

a) Xの位置に存在するならその節点に  
対する Transceternal 節点を返す

b) 存在しないなら、後方部分一致する近い節点が  
終点になるよう Transceternal 節点をつくり、  
二分木につけ加える

# 接点のトークン割り付けの再検討

---

**効果** 数バイト以上？ ソースコード中のループ要素が多いほど有効

**実装の手間** 手動でできる規模なら簡単、大規模では煩雑

**適用可能性** 中規模以上のプログラムならいつでも

ロシア文字などの 2 バイト文字を用いてソースコードを小さくすることについて述べた。

しかし、各トークンごとの出現回数はその節点「へ」繋がる接点の個数で決まり、出現回数はまちまちである。

トークンとして 1 バイト文字列を多用するほうがソースコードが小さくなることと合わせると、頻繁に出現する節点に 1 バイト文字を割当てると、2 バイト文字を割当てより小さくなる。

ex: ああ i は ii あ とすると 7 バイトから 5 バイトになる<sup>2</sup>

---

<sup>2</sup>この場合 1 バイト文字が余っているので、iic と全て 1 バイトトークンにすると 3 バイトになる

## 接点のトークン割り付けの再検討

---

実際には絶対アドレス指定用接点という大抵 1 枝のみ接続する接点が多多数を占めているので、2 回以上出現するトークンは可能な限り 1 バイト文字であるべき

生成器で自動的に行うなら、仮の状態で構築した Transceternal ソースコードに対して、出現頻度の高い順番に並べ、1 バイト文字列, ..., 2 バイト文字列 と順番に割り付けていく

手動なら `uniq -c` などを使用して頻度を調べ、使用頻度の高い多バイト文字ものと低い 1 バイト文字とを交換すればよい

# 変数配置の再検討

---

効果 中程度から高

実装の手間 煩雑

適用可能性 変数、あるいは変数アクセスアドレスが多彩な場合

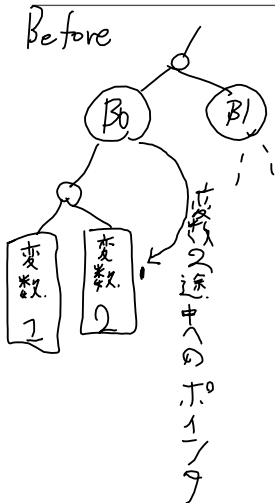
変数配置計画を再検討する

アクセスアドレスの suffix が多彩な場合ボトルネックになるので減らすのが有効

正しく動作しないときのデバッグや、多数の変数が絡みあうときの処理が手間なので実装は煩雑になる

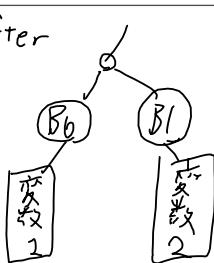
# 変数配置の再検討: 図解

Before

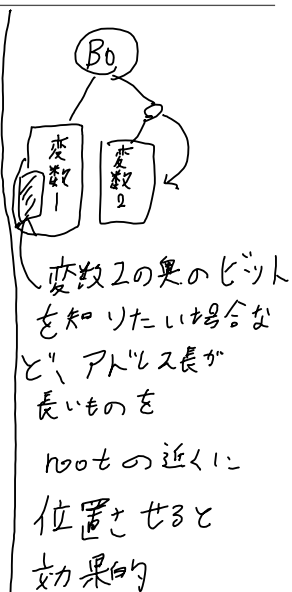


変数2へのアクセスは  
00000

After



変数1へのアクセスは  
00000



# 絶対アドレスの再検討

---

効果 場合によりけり

実装の手間 簡単なものから面倒なものまで

適用可能性 いつでも

先のものとも関係するが、絶対アドレスの suffix の多様性を減らすため、別の方策でアクセスできないか考える。

絶対アドレスの prefix が様々なものであっても、絶対アドレス指定用節点の共通部分は共有できるので節点数はさほど増加しないが、suffix に多様性がある (例: 1...10 と 1...11 ) と、全く共有できないので、共有できる場合と比べ倍程度容量がかさむ。

そこで、同一 suffix で全ての絶対アドレス指定部分を帰着できないか考えたり、特定の suffix の最長の長さを削減できないか考えるとよい<sup>3</sup>。

---

<sup>3</sup>なお、prefix 側の絶対アドレスを削減することも場合によっては有用。



基本的には問題の状況により異なるが、いくつかアイデアを挙げる

1. 1...1 (入力を見るような絶対アドレス指定) を沢山利用していて、そのうち 1 の個数が最長のものを一度しか使っていない場合は、「絶対アドレス指定のうち 1 の個数が少ない処理」で分割できないか考える。とくに 入力文字列を先に進めるなどとの処理で、一度に沢山のバイト列を移動させている場合 ( 1 に 111...1 を代入している場合) は複数に分割するとよい

2. 同じ節点を 別のリンクから見にいけないか検討する。似たようなアドレスに複数箇所からアクセスできる場合、たとえば入力の一部が変数に代入されている場合、変数が代入時から変更されていないと仮定できる場合は 1...1 を使いアクセスできないかと考えるとよい<sup>4</sup>。

---

<sup>4</sup>もし対象プログラムが入力をほとんど使わず変数への絶対アドレスを表す値を使っているのなら 1...1 をなるべく使わないような方針も必要かもしれない。

## 絶対アドレスの再検討: 図解

---

- 入力の特定bit が 0 かをみたい時:

絶対アドレス  $11 \cdots 111\underline{0}$

- 入力の特定bit 以降を複製したい時:  $11 \cdots 111\underline{1}$

2つのアドレス指定用節点(群)は 共有ではない

→ 片方で表せるようにする

# 絶対アドレスの再検討: 図解

## 例 リスト構造

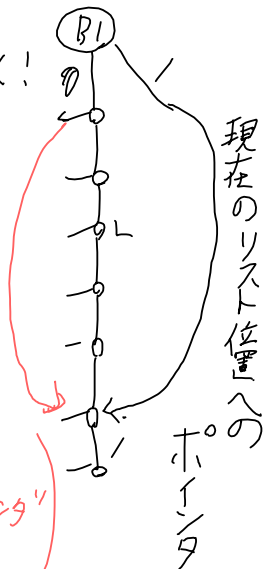
— リストの先頭へのアドレス。 浅く!

— リスト始点へのアドレス

の2つが必要であり, B1以下の貴重なアドレス領域を消費。

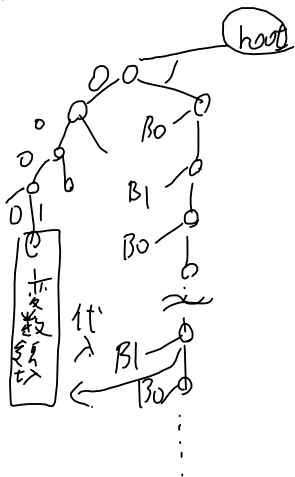
経験上, なるべく, リスト先頭アドレスを浅い階層においた方がよくなる

(赤字: 始点か1と確定すれば第1要素のローptrを“現在リスト位置ポインタ”にできないか? (未検証))



## 絶対アドレスの再検討: 図解

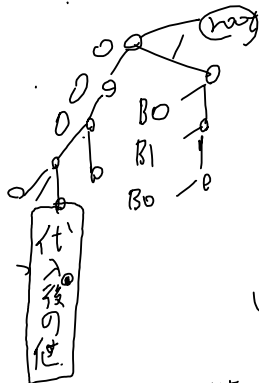
### 2. 変数代入



入力の一部を変数に代入  
した後、その変数の  
途中列にアクセスしたい

# 絶対アドレスの再検討: 図解

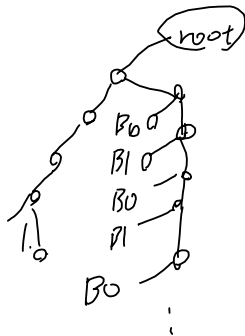
変数代入



代入後のアドレスで表す  
 00011...11  
 とアクセスすると

0 表現用ビット多数必要

vs.



代入前のアドレス  
 11...1

代入時にも使った  
 アドレス表現用ビットを共有

# 状態の一部共有

---

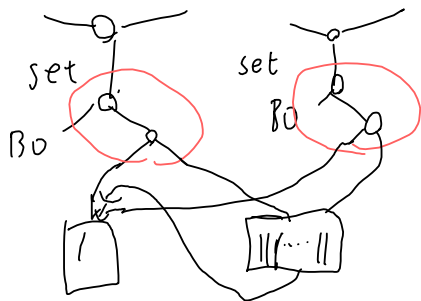
効果 数バイト?

実装の手間 見つけられれば簡単

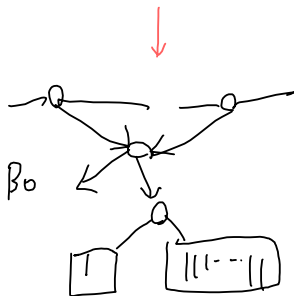
適用可能性 引数が2つとも一致している場合のみ。似たような処理が多い場合は可能性アリ

状態の2引数が一致している場合、状態を指定する部分以外のコマンドの節点を共通化すると削減できる

## 状態の一部共有: 図解



○ で "囲んだ"  
2 節点は  
0-pointer, 1-pointer だ  
と もに一致.



2 節点 削減

## B1, admin, root 接点を他の接点と共有

---

効果 数バイト？

実装の手間 高

適用可能性 変数が多くない小規模プログラムのみ。B1 と比べ  
admin は難しい

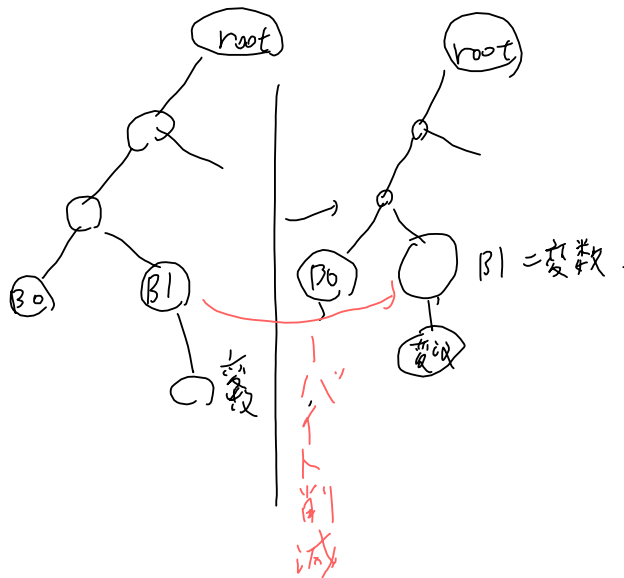
これは技巧的なので注意

admin に求められる条件は 0-pointer が B0, 1-pointer が B1 のみ  
であるので、条件を満たす節点があれば他にないなら他の節点で置き  
換えると 1 節点削減できる

また、B1 それ自体を変数の一部にすることもできる。実行されて  
いる間に B1 を指す節点が次々とかわっていくので、B1 を対象に  
した if や allocate が使えないことに注意。



## B1, admin, root 接点を他の接点と共有: 図解



問題点:

B1が実行途中で  
入れ替わるので

実行途中にB1を  
指すことができない

## 補足

何となく知っておくとソースコードの短縮に役立つかもしれない<sup>5</sup>  
直感

- ▶ Transceternal では 1 と 0 が非対称。B0 以外と B0 だと、B0 以外のほうが得やすい<sup>6</sup>
- ▶ デバッグ用にジェネレータの変更は小さくやっていくことが基本、一度に複数の変更をしない

コマンドごとの節点数消費量のイメージ。

- ▶ root, admin, B0, B1 で 4 接点
- ▶ set は 3 接点/コマンド
- ▶ allocate, if は 4 接点/コマンド
- ▶ アドレス指定用接点は 1 接点/0 or 1

---

<sup>5</sup>無駄な知識かもしれない

<sup>6</sup>01 以下のコマンド用節点など、アドレス指定と本来無関係の節点をアドレス指定用に転用することを試みると、1 の連続ばかりが得られる。B0/B1 の子を変数用に使っていると 0 が続くほうがいいのに難しい。

## 実践

トークンをマルチバイト文字にしてトークン区切りの空白をなくす。1560 バイト → 900 バイト

0122245363738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3o3p3q3r3s  
3t3u3v3w3x3y3z3A3B3C2D3E3F  
0 0 0 0 0 0 ζ 3 η 3 θ 3 ι 3 κ 3 λ 3 μ 3 ν 3 ξ 3 ο 2 ο  
ή 3 ί 3 ú 3 α 3 β 3 γ 3 δ 3 ε 3 ζ 0 0 0 0 0 0 0 0 ε é 3  
ή 4 5 7 8 9 α δ α ε θ Γ δ ζ F F γ ÿ 3 á ȳ φ ω  
π ρ ϕ í 3 Ÿ Ʒ Ɔ 9 θ 9 Γ 7 α Ω 3 í ü ó ó ú ú ú Ű  
3 Ω φ χ χ χ ψ ω Ω α π ρ σ ς σ τ X 3 Ψ í EU2Tf  
2G2H3I3J3K3L3M3N3O3P3Q3R3S  
3S 0 6 2 a 0 3 0 0 3 и 3 р 3 с 3 j 3 Ө 3 ε 3 э 3 Р 3  
р 3 с 3 М 3 м 3 р 3 О 3 С 3 о 3 È 3 È 3 Ъ 3 Ѓ 3 Е 3 S  
3 I 3 I 3 J 3 Љ 3 Њ 3 Ћ 3 Ќ 3 Ў 3 Ў 3 Ў 3 Ў 3 Ў 3 Ў 3  
Г 3 Д 3 Е 3 Ж 3 З 3 И 3 Й 3 К 3 Л 3 М 3 Н 3 О 3 П 3 Р  
3 С 3 Т 3 У 3 Ф 3 Х 3 Ц 3 Ч 3 Ш 3 Щ 3 Ъ 3 Ы 3 Ь 3 Э 3  
Ю 3 Я 3 Я в ф 2 у г 2 д 2 е 2 ж 2 ж з 2 и 2 й 3 к 3 л  
3 м 3 н 3 о 3 п 3 р 3 с 3 т 3 т х е 2 ф ц 2 ч 2 ш 3 щ 3  
ъ 3 ы ы 2 ь 2 э 2 ю 3 я 2 è 2 ë 2 ħ 2 ħ s 2 ħ i 2 ħ  
2 j 3 ь 2 ъ 2 ħ 3 ħ k 2 ħ 2 Ÿ 3 u 2 Ű 2 w 3 Ъ 3 Ъ o y Q  
o o o Ű Ű Ű 2 P 2 p 2 ψ 2 ø 2 V 3 v 2 V 2 v 3 O y 2  
O y 2 2 0 0 0 2 0 2 0 0 2 0 3 0 0 2 v φ 2 Y w  
2X2Y3Z3I3!"#3\$3%3&3'3(3)3\*3+3,-3-3./3:3;3<3=3>3?3@3[3  
3\3]3^3\_3`3{3|3}3~3Σ 3 Τ 3 Τ 0 VVVVVVVV

## 実践

1...1, 1...10 の絶対アドレス指定を共有して、入力の 8bit のうち、実際に判定する 4bit 分のみを比較する。900 バイト → 553 バイト

[illegible]

# 実践

---

走査対象を 8 文字ずらすのに、一度に 64bit 進めていたものを 32bit ずらす処理 2 回に分ける。1...1 のアドレス指定用節点が 65 個必要だったのが 33 個に減少。466 バイト<sup>7</sup>

```
012223463738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3o3p
3q3r3s3t3u3v3w3x3y3z3A3B3C3525D32
D é ħ ó í ú um Ω Ĭ ō Ÿ á t l Y Φ 0 X Ψ sk}
-0 Σ T r j _ ` 0 { | ħ p [ \ 0 ] ^ f n E U 2 T F 2 G 2 H 3 I 3 J 3 K
3 L 3 M 3 N 3 O 3 P 3 Q 3 R 3 S 3 S α γ 2
β SW3X3Y3Z3!3"3#3$3%3&3'3(
3)3*3+3,3-3.3/3:3;3<3=3
>3K δ ζ 2 ε SW η ω 2 ψ θ 2 ι 2 κ 3 λ 3 λ μ
2 ν 2 ξ 3 ο 3 π 3 ρ 3 ς 3 σ 3 τ 3 υ 3 φ 3 χ 3
χ ĭ Ÿ 2 γ F ü 2 ó 2 ú 2 ő 3 K 2 θ 2 θ 2 γ 2 γ
φ □ 2 □ ω 2 κ 2 Q 3 ρ 2 Γ 2 ς 3 ς F 2 F 2 ħ 3
ς 2 ϭ 2 ϭ 3 □ 3 □ p C 0 M M □ 2 □ 2 κ 2 κ ρ 2
c 2 j 3 Θ 2 ε 2 ε 3 P 2 P □ □ 2 □ S □ 2 □ 2 □
3 □ 2 □ 3 □ 3 □ 2 é V@2?FW α VVVVV
```

---

<sup>7</sup>手動で 2 回以上登場する 2 バイトトークンと 1 回しか登場しない 1 バイトトークンを置き換えると 456 バイト

変数配置の再検討、32 個ある入力をかぞえていたループ回数カウンタを B0 直下に移動。421 バイト

```
01226346C32C738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3
o3p3q3r3s3t3u3v3w3x3y3z3
A3B35253 á é ò í ï t1 Ψ Ω ö ÿ sk
T Y 0 Φ X rj|}0-Σ qi^_0`{go@
[0\]emDT2SE2F2G3H3I3J3K
3L3M3N3O3P3Q3R3R ú ß 2 α RV3W3X3Y3Z3!3"3#3$3%3&3'3(
3)3*3+3,3-3.3/3:3;3<3=3
J γ ε 2 δ RV ζ ψ 2 χ η 2 θ 2 ι 3 κ 3 λ 2 μ
2 ν 3 ξ 3 ο 3 π 3 ρ 3 σ 3 τ 3 υ 3 φ 3 ϖ ω
ώ 2 ú Ē ĩ 2 ü 2 ó 2 ő Ɔ 2 ρ 6 2 θ 2 Υ 2 Ƴ 3
Υ Ỳ 2 ϕ 2 Ƶ 2 Ʒ 3 Ɔ 3 Ɔ 0 0 0 ĩ 2 2 2
2 2 3 0 2 2 ç 2 2 R F 2 F 2 Ƨ 3 4 2 Ƨ 3 Ƨ 3
Ƨ 2 á U?2>EV Ű UUUUU
```

# 実践

---

変数位置の検討、出力保存用変数の位置を移動し、B1 そのものを出力保存用変数の起点にする。このことでもはや B1 を指す節点は固定ではなく可変。421 バイト → 403 バイト

```
0123325363738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3o3
p3q3r3s3t3u3v3w3x3y3z3A
34243 Ω Ĩ 0 Ÿ á sk Y Φ 0 X Ψ rj}-0 Σ
T qi_`0{ |ph[\0]^fn=>0?@dlBQ2PC2D2E3F3FG3H3I3J3K
3L3M3N3O30 é í 2 ħ 0S3T3U3V3W3X3Y3Z3!3"3#3$3%3&3'3(
3)3*3+3,3-3.3/3:3G ú ß 2 α 0
S γ σ 2 ς δ 2 ε 2 ζ 3 ζ η 2 θ 2 ι 3 κ 3 λ 3 μ
3 ν 3 ξ 3 ο 3 π 3 ρ 3 ρ τ ω 2 ψ C υ 2 φ 2 χ 2
χ ï φ 2 Ÿ ü 2 ó 2 ú 2 ó 3 ó κ 2 β 2 θ 2 Υ 3 Υ
3 Ξ □ 0 □ □ u ჯ 2 ჯ 2 ჯ 3 □ 2 □ ∞ F 2 F
0 κ 2 Q 2 q 2 T 2 c 3 c 2 Ω R<2;CS é RR
RRR
```

# 実践

---

入力 32 個を数える ループ回数カウンタは初期値に 1....10 (1 は 32 個続く) を表す 33 個の節点を作っていたが、1....1 (いろいろなところで使われている!) で表現できることに気づく。1...10 を表す アドレス用節点が削減できる。403 バイト → 363 バイト

```
012332435363738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3
o3p3q3r3s3t3u3v3w3x3y3z323
}-0 Σ T "3#3$3%3&3'3(3)3*3L2L
T3U3V3W3X3Y3Z3!3" _`0{!|S3T[\0]~ZR3S=>0?@
YQ3R/;0;<03P3QW+,0-.M3N3OUAG2FB2C2D3E3E
r Y X 2 Φ z I 34 Ψ Ĩ 2 Ω z I Ÿ λ 2 κ á 2 é 2 ħ 3 ħ
í 2 Ů 2 α 3 β 3 γ 3 δ 3 ε 3 ζ 3 η 3 θ 3 ι 3 ι μ ρ 2 π
B v 2 ξ 2 ο 2 ο ς ú 2 ó σ 2 τ 2 υ 2 φ 3 φ χ 2 ψ 2 ω 2
ĩ 3 ü 3 ü ɰ 0 Q ϕ v σ ó ϕ 2 Ÿ z K 2 β 2 θ 2 Υ 2 Υ
3 ˘ 2}HK2JBI Y HHHHH
```



# 実践

---

無駄な if 文が残っていたのを減らす。処理開始時に、2 文字目=3 文字目をチェックするときに、最初 1 文字目 1bit にいる状態で 2 文字目を見るのではなく、先頭が 2 文字目になるよう、最初に移動コマンドを加えた。(各ビットをチェックするためのアドレス指定用節点) 111...10 の 1 の個数を 24 個から 12 個に減らす。310 バイト<sup>8</sup>

012332435363738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3  
o3p3q3r3s3t3u3v3w3x3y3z323KM2Lzr-.0/:Z3N2NR3S3  
T3U3V3W3X3Y3Z)\*0+,YQ3R%&0'(XP3Q!"0#\$W03PAG2FB2  
C2D3E3Ez;>2=z<34?[2@z<\| 2Ω]2^2\_3\_`2{2|3}3~3Σ 3T  
3Y 3Φ 3X 3Ψ 3Ψÿ Ů 2í B á 2é 2ή 2ή α μ 2  
λ β 2γ 2δ 2ε 3ε ζ 2η 2θ 2ι 3κ 3κ υ φ 0χ  
ψ á β ν τ 2σ z ξ 2ο 2π 2ρ 2ς 3ς 2-HJ2IBb;HHH

---

<sup>8</sup>手動で 2 回以上登場する 2 バイトトークンと 1 回しか登場しない 1 バイトトークンを置き換えると 304 バイト

# 実践

---

出力を置いている場所の仮ポインタの変数位置を検討。285 バイト<sup>9</sup>

012333435363738393a3b3c3d3e3f3g3h3i3j3k3l3m3n3  
o3p3q3r3s3t3u3v3w3x3y3z323KM2Lzr-.0/:Z3N2NR3S3  
T3U3V3W3X3Y3Z)\*0+,YQ3R%&0'(XP3Q!"0#\$W03PAG2FB2  
C2D3E3Ez;@2?<2=2>3>s^{2`z\_34  
|~2}z\_Σ Ψ 2X BT 2Y 2Φ 2Φ Ωδ 2γ ï 2ÿ 2ά 2έ  
3έ ή 2ί 2ú 2α 3β 3β ν ξ 0ο π Τ ï ε μ 2λ zζ 2η 2  
θ 2ι 2κ 3κ 2-HJ2IBb[]2\<4^HHH

---

<sup>9</sup>手動で2回以上登場する2バイトトークンと1回しか登場しない1バイトトークンを置き換えると280バイト

# 実践

---

00011 と 0011 を指すアドレス指定用節点は 0011 部分が共有できるが、今まで忘れていたので共有。250 バイト

012333435363738393a3b3c3d3e3f3g3h3i3j3k3l  
3m3n3o3p3q3r3s3t3u3v3w3x3y3z323KM2Lzr-.0/  
:Y3Z2ZQ3R3S3T3U3V3W3X3Y)\*0+,XP3Q%&0'(W03P  
!"0#\$VN30AG2FB2C2D3E3Ez;@2?=2>2<3<s^{2`z\_  
34|~2}z\_Σ Ψ 2X BT 2Y 2Φ 2Φ İ é  
2ά Ψ 2=Ω 2B β γ 0δ ε T Ψ ή α 2Ú  
zı 2Ψ 2-HJ2IBb[]2\=4^HHH

00011 と 00001 の最後の 1 バイトを共有して 240 バイト<sup>10</sup> 走査対象を 8 文字ずらすのに、一度に 64bit 進めていたものを 32bit ずらす処理 2 回に分けたとき、2 回とも引数は同一なので、次状態へのポインタ以外は統合できる。233 バイト  
トークンの 3 度出現する 2 バイト文字を 1 度しか出現しない 1 バイト文字に入れ替えて 231 バイト

---

<sup>10</sup> 統合して 237 バイト

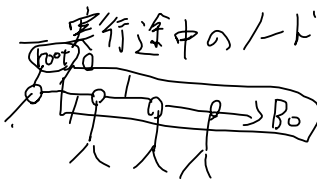
## おまけ

---

以下では没となったアイデアを取り上げる。場合によっては利用できるかもしれないが、過度な期待は禁物

— "B0以外" という条件を使いこなす

— 実行途中のノード そのものをタイマーとして利用



← 実行するステップごとに1減少  
プログラム・カウンタとして利用できる

# 動的命令生成

---

今回は命令は全て静的に生成していたが、起動後に動的に生成できないか？

問題点: `allocate` 命令をつかうために 4 接点必要なこと、ソースコードが読みにくくなること

`allocate` のコストが高く、今のところ利用可能な場面に思い至らない

## ブックマーク (仮)

---

deserialize 対象について、接点が 1 と判定される基準は 0-pointer が B0 でないこと。B1 でなくてもよい。

“B0 以外” の接点を今までは B1 などと定めていたが、そこに情報を埋め込み、深い階層にある変数を文字数の少ない絶対アドレス表現で表すことを期待する

有益な接点にアクセスするための絶対アドレスが deserialize 対象の絶対アドレス + 最初の 1 までのアドレス数 + 0 の 5 節点? 程度で表現できる。

今回は変数の個数が少なく、全ての変数を B0, B1 直下に配置したので、ブックマークせずとも元々の位置で既に 4 バイト以下でアクセスできた。この方針では有益にならなかった。

## ブックマーク (仮)

---

if 扱いになる条件は また、B0 でも B1 でもない接点を指すときであった

その接点を新しく作らずに既存接点 (root) にすることで if に必要な接点が 1 つ省略できるのは紹介した通りだが、そこに情報を埋め込、

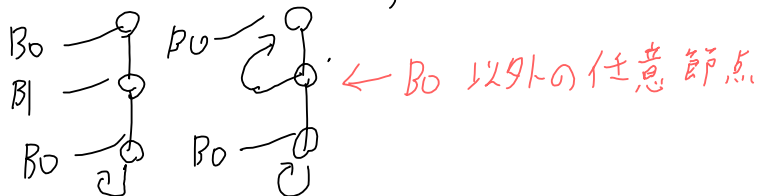
プログラム中実行中に if 対象の接点は絶対アドレス “0100” を表す値のたった 8 バイトでアクセスできる。深い階層にある変数を if 中の左辺や右辺に使うとき、その変数の接点へのポインタを “0100” に置いておくと 必要な絶対アドレス表現の長さが削減できる。

今回は変数の個数が少なく、全ての変数を B0, B1 直下に配置したので、ブックマークせずとも元々の位置で既に 4 バイト以下でアクセスできた。この方針では有益にならなかった。



# ブックマーク (仮): 図解

deserialize 基準を考えると, 以下の2つは同一



0/0

0/0

