「今度こそわかるＰ≠ＮＰ」（渡辺治）の補足説明
講談社サイエンティフィク, 2014
(2014.3.9)

原始計算機械・原始プログラムの補足（タイプ２）

原始プログラムの例を見て,「通常の機械語が大分単純になったものかな」と思ったのだけれども,「ちょっと待てよ，でも何だか少し違うなぁ」と感じた人たちもいるだろう．ここでは，その人たち向けの補足を述べる．なお，用語や記号は本書のものを適宜用いる．

確かに，原始計算機械は普通のコンピュータに似ている．大分，普通のコンピュータを大分単純化したもの，と思える．けれども，次の点で異なっている．

(1) 各番地のメモリが格納できるデータは 1 ビットしかない．
(2) レジスタ長に制限がない．
(3) 制限レジスタ長や制限時間という概念が入ってきている．
(4) 入力命令がない．（ちなみに，出力は停止命令 `halt` で指定する．）
(5) 掛け算や割り算が `mult2` のように限定されている．レジスタが 4 つ（しかない or も有る！）

こうした違いに違和感を覚えた人も多いだろう．以下で，なぜそのように決めたのかを説明する．

(1), (2) メモリとレジスタ

普通のコンピュータではメモリ（記憶領域）はバイト (8 ビット）単位で管理されている．ここではそれを単純化して 1 ビット単位にした．一方，普通のコンピュータではレジスタ長は固定である．たとえば，32 ビットマシンとか，64 ビットマシンなどがある．これは機械の制約から決まる．

このレジスタ長は使用できるメモリの範囲も決めてしまう．レジスタはメモリの番地を格納するためにも使われるからだ．32 ビットマシンが使える番地は（通常は）$2^{31}-1$ くらいまでである．一方，我々は（理論上）入力長に制限を設けたくない．1 つのプログラム（原始計算機械）は，どんな長さの入力に対しても処理できると考えたい．そのためにはメモリ空間も（入力長に応じて）十分大きくとれるようにしたい．そこでレジスタ長には制限を設けなかったのである．

ただし，プログラムがさわれるメモリの数は高々計算ステップ数分だけだ．10 個の命令の実行で 10 箇所より多くのメモリはさわれない．一方，メモリは計算開始時に 0 にセットされているものとしている．ということは使用するメモリの数だけ初期設定に時間がかかることになる．という訳で「番地の値は制限計算時間以下」という使用上の制限を設けることは妥当だろう．たとえば制限計算時間が 2000（ステップ）ならば，使用できるメモリの番地は $0 \sim 1999$ まで，それを表わすには $\lceil \log_2 2000 \rceil + 1$ ビットあれば十分だ．ということでレジスタ長 $l(\ell)$ を制限時間 $t(\ell)$ に対して $\log_2 t(\ell)$ 程度（余裕を持って $2\log_2 t(\ell)$）と仮定してもメモリ管理上は問題は生じない．

(3), (4) 原始計算機械の動かし方

本書では，原始計算機械を動かす上で「制限計算時間」と「制限レジスタ長」というパラメータ（＝数）を用いることにした．原始計算機械が（与えられた入力例に対する）計算で使用してよいステップ数（＝実行命令数）を決めるのが制限計算時間で，レジスタのビット数を決めるのが制限レジスタ長である．これらは原始計算機械の固定レジスタに実行開始時に格納され，計算途中でステップ数やレジスタ長が，これらの値を超えると異常終了することになっている．

なお，制限計算時間や制限レジスタ長の値は入力長に応じて決まる．決め方は各原始計算機械ごとに好き

な関数（ただし妥当なもの）を用いてよいことにする．

まず，なぜこのような制限を設けたのか？そもそもプログラムによっては入力長だけからでは計算時間が決まらないものもある．計算時間が計算不可能なプログラムさえ作ることもできる．

計算複雑さの理論では（ほとんどの場合）そのような病的なプログラムを使う必要はない．計算ステップ数（の妥当な上界）を入力長 ℓ に対して，$12\ell^2+52$ とか $3\ell\log\ell+212$ などといったような簡単な関数で抑えられるプログラムしか考えなくても問題が生じないのだ．そのため計算複雑さの理論では，議論をやりやすくするために「時間限定機械」を用いることが多い．今回もその流儀に従ったのである．

では，ある原始計算機械 M （仮に制限計算時間を決める関数は $12\ell^2+52$ であるとする）に対し，実際に入力列 010010 を与えて実行する際，誰が $12\times 6^2+5=537$ を求めて定数レジスタにセットするのだろうか？また，入力列をメモリの 0 番地から 5 番地にセットするのは誰なのだろうか？本書では「そういう準備は誰かがやってくれる」という，非常に楽観的な考え方をとる．その議論を省略したかったからだ．本来は入力列は入力命令を使ってメモリに読み込む．そして，その長さを数えて，制限計算時間を求め，実行ステップ数がそれを超えないように監視しながら計算する，そのようにプログラムを作るべきだが，その部分は省略してしまったのである．

(5) 基本演算の選び方

掛け算（割り算も）は特殊なもの（つまり，2 倍する計算のみ）しか基本演算には許していない．これは 1 ステップの命令で直観的には定数ステップでできそうもないことを行ってしまうのを防ぐためである．「掛け算なんて普通のコンピュータでは基本命令として用意されていますよ！」と思われる人もいるだろう．それはレジスタ長が固定だから可能なのである．

我々の計算モデルのようにレジスタ長に制限がないとどうなるか？たとえば 1000 ビットの数同士の掛け算が 1 命令でできてしまうことになる．一方，2 倍するだけならば，最下位ビットに 0 を 1 つ加えるだけなので（やりようによっては）定数ステップでも実現できそうだ．

「それならば加算だって状況は同じじゃないの？」と思われた人はするどい！確かに，足し算だって 1000 ビットの数同士の足し算は定数ステップ数でできそうにない．ではなぜ足し算がよくて掛け算がいけないのか？

それは掛け算は列の長さ（2 進数の桁数）を倍増させるからだ．たとえば，2 乗を 20 回繰り返しただけで，1 ビットの数が 1 Mbit (100 万ビット）にもなってしまう．そういう計算を許すと，それを使って指数時間の計算も多項式時間で模倣できてしまうのだ．一方，足し算は長さが高々 1 ビット増えるだけ．確かに定数ステップではできないかもしれないが，まあ，多項式時間を議論するには影響ないので，大目に見たのである．（じゃぁ「それが TIME[ℓ^2] の意味を大きく変えるか否か？」と聞かれると，ちょっと困ってしまいますね．多分，少し違うでしょう．）

補足説明 ot2-compmodel_type2 **終了**