「今度こそわかるＰ≠ＮＰ」（渡辺治）の「こだわり補足」

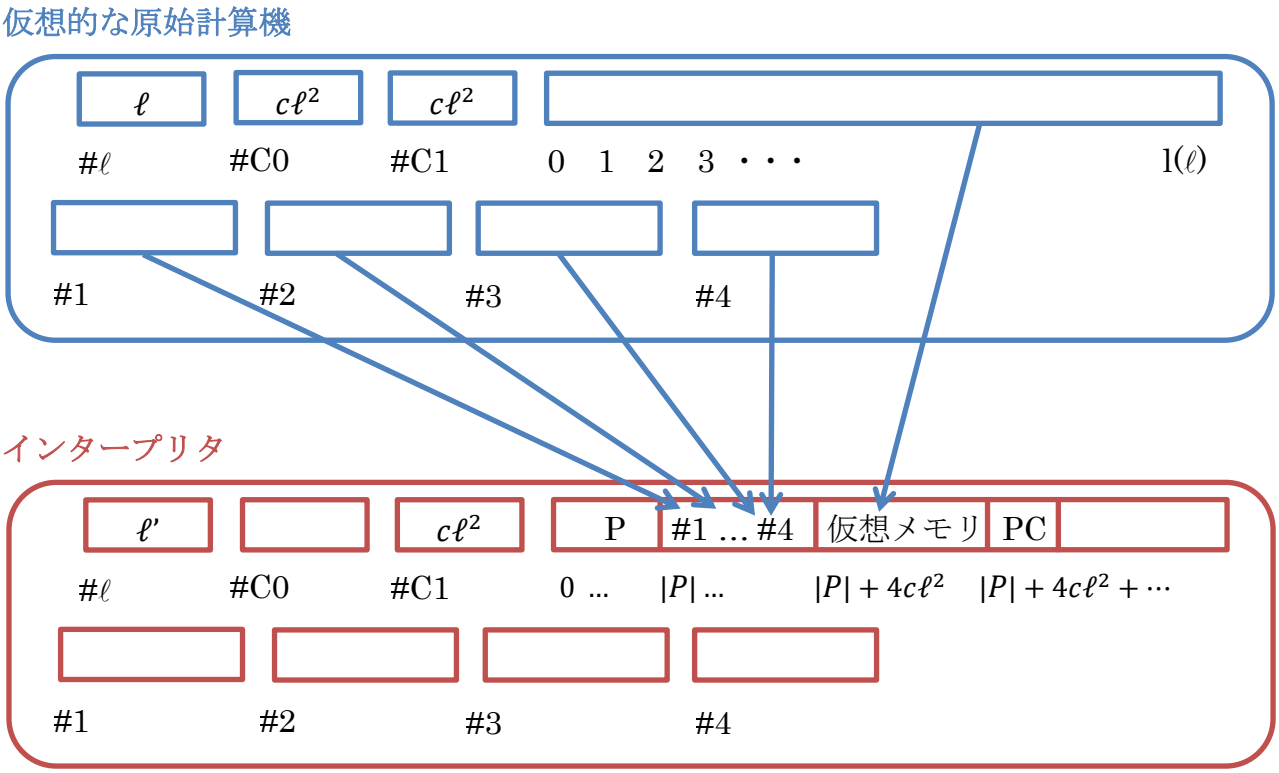
講談社サイエンティフィク, 2014

(2014.9.18)

原始プログラムのインタープリタ

※この説明は，東京工業大学で大学院 授業課題の回答レポートを元に作成した．元となったレポートを作成し加藤 大智氏（2013 年度計算工学修士）に感謝する．

原始プログラムのコード P とそのプログラムへ与えられる入力 x が与えられたとき，その実行をシミュレーション（模倣）するインタープリタの構成法の概略を以下に示す．実際のプログラムは，本ページに紹介しているインタープリタ（Java で記述）を参考にして欲しい．



図：インタープリタと仮想的な原始計算機の関係

図のように，仮想的なレジスタとメモリを原始計算機のメモリ上に置くことで，原始計算機上でインタープリタを実現する．

先に述べた P のコードを原始計算機のメモリの番地 $[0, \dots, |P| - 1]$ に格納する．また仮想レジスタとして $[|P|, \dots, |P| + 4c\ell^2 - 1]$ を，仮想メモリとして $[|P| + 4c\ell^2, \dots, |P| + 5c\ell^2 - 1]$ を，プログラムカウンタ（PC）として $[|P| + 5c\ell^2, \dots, |P| + 5c\ell^2 + \log|P|]$ を確保する．PC の値は，その時点で実行しているコードの実メモリ

上の位置である．例えば P の i 行目を実行しているなら $i(8+b)$ とする． PC の値は高々$|P|$なので，PC の容量は $\log|P|$ で十分である．通常の原始計算機と同様に，問題例 x は実行開始時に仮想メモリに格納される．

P の i 行目の命令は次のようにして実行される．なお fetch(#a, #b, l)とは，メモリ番号 #b から lビット分の値を #a に取得するプログラムである．本書（こだわり２；ただし訂正あり）でも説明した@fetch の#1 を#a，#2 を#b，#c1 を#c に変更したものに等しい．store も本書にあるものを使う．

1. プログラムカウンタをレジスタ#4 に fetch(#3, $k(8+b)+5c\ell^2, \log|P|$)，
2. メモリ　$[i(8+b), \ldots, i(8+b)+5]$　をレジスタ#1 に fetch(#1，#4, 6)
3. sub#1 と 0eq?#1 を繰り返し i 番目の命令がなにかを特定
4. 命令がレジスタの値を必要としている場合，その値を仮想レジスタからレジスタ#1 に fetch(#1，$|P|+rc\ell^2$，$c\ell^2$)（ただし r はレジスタ番号−1）
   レジスタ#1 の値はもう必要ないので上書きしてしまってもよい．
5. 命令が目的語を必要としている場合，目的語をレジスタ#2 に読み込む．$[i(8+b)+8, i(8+b)+9]$ を fetch(#2，$\#4+6$，2)する．これは目的語をどう解釈するかの値（前の原始プログラムのコード化法参照）．
   5.1. #2 = 0 なら目的語の値を fetch(#2，$\#4+8$，b)．（つまり，目的語の値をレジスタ#2 に fetch する．）
   5.2. #2 = 1 なら目的語の値を fetch(#2，$\#4+8$，b)し，$\#4:=|P|+(\#2-1)c\ell^2$という計算で仮想レジスタの最初の番号を#4 に入れ，レジスタ#2 に fetch(#2，#4，$c\ell^2$)．（つまり，目的語で指定された番号のレジスタの値を仮想レジスタから fetch する．）
   5.3. #2 = 2 なら目的語の値を fetch(#2，$\#4+8$，b)し，$\#4:=|P|+4c\ell^2+\#2$という計算で仮想メモリの番号を#4 に入れ，レジスタ#2 に fetch(#2，#4，$c\ell^2$)．（目的語で指定された番号のメモリの値を仮想メモリから fetch する．）
   5.4. #2 = 3 なら目的語の値を fetch(#2，$\#4+8$，b)し，$\#4:=|P|+(\#2-1)c\ell^2$という計算で仮想レジスタの最初の番号を#4 に入れ，レジスタ#2 に fetch(#2，#4，$c\ell^2$)．$\#4:=|P|+4c\ell^2+\#2$で得られた仮想メモリの番号を#4 に入れ，レジスタ#2 に fetch(#2，#4，$c\ell^2$)．（つまり，目的語で指定された番号のレジスタの値を仮想メモリから fetch し，その番号のメモリの値を仮想メモリから fetch する．）なお，レジスタ#4 の値は 8. で再度読み出すので上書きしてしまってもよい．

6. 命令で指定された計算を行う．ただし命令で指定されているレジスタは#1 に，目的語は#2 の値にそれぞれ読み替える．
7. 計算結果を命令で指定された仮想レジスタに store する．
8. プログラムカウンタを fetch し$(8+b)$加えて store する．ただし 0eq? や true? が真だった場合は$2(8+b)$加え, 命令が goto だった場合は目的語の値×$(8+b)$に変更する．

さて必要なステップ数だが, 1 回の fetch は$3+6l$ ステップ, 1 回の store は $1+10l$ ステップ, レジスタ上の数同士の任意のかけ算#1*#2 は 5*#2 ステップで実行できる．また命令セットのサイズや $|a|$ はあらかじめ与えることが可能な定数であるとする．したがって各段階での最悪ステップ数は,

1. は定数$+6\log|P|$ステップ
2. は 39 ステップ
3. は 128 ステップ
4. は$6c\ell^2$ステップ
5. は定数$+6b+5c\ell^2+6c\ell^2+5c\ell^2+6c\ell^2=6b+22c\ell^2$ステップ（5. 4 のとき）
6. は 1 ステップ
7. は$10c\ell^2$ステップ
8. は$4+b+10c\ell^2$ステップ

である．以上を足し合わせると，1 回の命令の実行に 定数$+6\log|P|+7b+48c\ell^2<61|P|c\ell^2$ ステップ必要となり， $c\ell^2$ 回の命令を実行する場合 $61|P|c\ell^2\times c\ell^2=61|P|c^2\ell^4$ 時間で実行可能である．