プログラムの命令列表現の自由度に関する研究

指導教官：市川周一　　学籍番号：013740 八反田一宏

1　背景と目的

プログラムは命令列として実現されるが，命令列表現は一意に定まらない．同じ機能を実現することができれば，どのような命令列でもユーザにとって等価である．本研究では，このように同じ機能を実現する命令列を「機能等価な命令列」と呼び，1 つのプログラムに対する機能等価な命令列の数を「命令列表現の自由度」と呼ぶ．このような自由度は電子透かしや情報隠蔽に利用できる [1]．

本研究の目的は，命令列中の命令とデータの位置を変更することで得られる自由度を定量的に測定することである．しかし機能等価な命令列を得る方法は多数あるため，全ての自由度を数え上げることは難しい．そこで本研究では，(1) グローバル変数の配置順序，(2) ローカル変数の配置順序，(3) 基本ブロックの配置順序，(4) 基本ブロック内の命令の配置順序，の 4 種に関して自由度を測定する．さらにベンチマークプログラムを用いて，配置順序変更前後の実行性能とサイズの変化を測定する．全ての測定は Intel 命令セット (ELF 形式) のオブジェクトで行い，オブジェクト生成には GCC 2.95.3 と bintuils 2.13 を用いた．

2　変数の配置順序

一般にユーザはプログラム内の変数のアドレスを意識しない．従って (複数の) 変数のメモリ上の配置順序を入れ替えても，プログラムは機能等価になる．n 個の変数があれば $n!$ の配置自由度が存在し，これにより $n!$ 通りの機能等価な命令列を生成できる．

C 言語の変数には，グローバル変数とローカル変数がある．さらにグローバル変数は，外部変数，static 変数，初期化済み変数に分類できる．外部変数のアドレスはリンカ (ld) が決定するので，リンカのアドレス決定部を改造すれば，外部変数の配置順序を変更できる．static 変数と初期化済み変数は，それぞれ.bss と.data セクションに登録されるので，コンパイル後にアセンブリ言語ファイル中の定義順序を変更することにより，配置順序を変更できる．

ローカル変数はスタック上またはレジスタ上に配置される．スタック上のローカル変数は，ebp レジスタとオフセット値を用いて参照される．このオフセット値はコンパイラが決定するため，コンパイラのアドレス決定部を改造することにより，スタック上のローカル変数の配置順序を変更できる．レジスタ上のローカル変数に関しても，レジスタ割り当てによる自由度が存在するが，本研究では扱わない．

3　命令の配置順序

命令列は基本ブロックに分割することができる．基本ブロックとは，先頭に制御が与えられた時，途中で分岐せずに終端で制御の離れる命令列である．実行順序を適切に保持すれば，基本ブロックの配置順序を変更しても機能は等価になる (図 1)．この性質を利用してプログラムに電子署名を埋め込むことができる [2][3]．本研究では，アセンブリ言語ファイル中の命令列を基本ブロックに分割し，順序変更を行うプログラムを作成して，配置順序の変更を行った．

さらに基本ブロック内の命令も，命令間の依存関係がなければ配置順序を変更することができる．例えば図 2 において，命令 (1) と命令 (2) の順序を変更しても機能等価になる．一般に，命令 x の参照要素集合を $Ref[x]$，格納要素集合を $Sto[x]$ としたとき，$(Ref[x] \cap Sto[y]) \cup (Sto[x] \cap Ref[y]) \cup (Sto[x] \cap Sto[y]) = \phi$ であれば，命令 x, y 間に依存関係はないといえる．本研究では，アセンブリ言語ファイルを入力とし，基本ブロック毎に機能等価な命令配列を数えるプログラムを開発した．

4　実験と結果

本研究では，測定対象として C 言語のベンチマークプログラムとフリーソフトウェアを使用した (表 1)．n 要素の配置自由度は $n! = O(n^n)$ となり，n が大きくなると処理が困難になるので，今回は PPS (Partial Permutation Scheme)[4] を採用して実装した．要素は 6 個ごとのチャンクに分割し，各チャンクが $6! = 720$ 通り

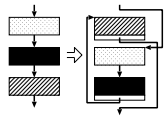

図 1: 基本ブロックの順序

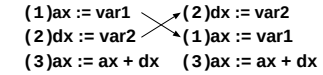

図 2: 命令の順序

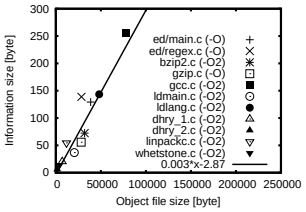

図 3: 基本ブロックの順序による埋め込み情報量

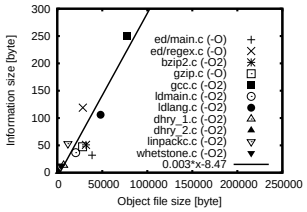

図 4: 基本ブロック内の命令順序による埋め込み情報量

の自由度を持つと計算する．端数となる要素は切り捨てて，測定から除外した．

グローバル変数，ローカル変数，基本ブロック，基本ブロック内命令の配置順序について，それぞれ自由度を測定した結果を表 1 にまとめる．一般に変数の数より命令数の方が大きいため，基本ブロックの順序および命令の順序から得られる自由度が大きくなっている．特にローカル変数については，コンパイル時に最適化を行うと多くの変数がレジスタ上に割り当てられるため，メモリ上での配置自由度が失われてしまう．本研究ではレジスタ配置の自由度を測定していないので，本件に関しては今後の課題とする．

基本ブロック配置と基本ブロック内命令配置の自由度を情報量に換算し，埋め込み可能情報量とオブジェクトファイルサイズの関係を調べた．結果は図 3 と図 4 に示す通りで，いずれも相関関係は明らかである．最小二乗法で近似した結果，いずれの場合も情報密度 (ファイルサイズあたりの埋め込み可能情報量) は約 0.3%であった．グローバル変数を用いた埋め込みにも一定の相関関係が認められ，その場合の情報密度は約 0.02%であった．

上記各手法では命令やデータの配置を変更するため，実行性能やオブジェクトサイズに影響が現れる可能性がある．そこで 3 種のベンチマークプログラム (dhrystone, linpack, whetstone) について，元のプログラムを 100 回実行した平均性能値と，ランダムに配置順序変更を行ったプログラム 100 個の平均性能値を測定した．測定には Xeon 2.80GHz，RedHat Linux 9 を用いた．基本ブロックの順序変更では，実行順序保持のためにジャンプ命令を挿入する必要があるため，性能は最大 6.1%低下した．基本ブロック内の命令順序変更，グローバル変数の順序変更，およびローカル変数の順序変更では，それぞれ性能は最大 3.1%, 1.7%, 1.5%低下に留まった．

これらのベンチマークプログラムに関して，オブジェクトファイルサイズの変化も調べた．グローバル変数，ローカル変数の順序変更，および基本ブロック内の命令順序を入れ替えた場合には変化はなかった．基本ブロックの順序変更を入れ替えた場合のみ，最大 5.1%サイズが増加した．

参考文献

[1] Collberg, C. S. and Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection, *IEEE Trans. Software Eng.*, Vol. 28, No. 8, pp. 735–746 (2002).

[2] Davidson, R. I. and Myhrvold, N.: Method and system for generating and auditing a signature for generating and auditing a signature for a computer program, *USPatent*, No. 5,559,884 (1996).

[3] Hattanda, K. and Ichikawa, S.: The Evaluation of Davidson's Digital Signature Scheme, *IEICE Trans. Fundamentals*, Vol. E87-A, No. 1, pp. 224–225 (2004).

[4] Ichikawa, S., Chiyama, H. and Akabane, K.: Redundancy in 3D Polygon Models and Its Application to Digital Signature, *Journal of WSCG*, Vol. 10, No. 1, pp. 225–232 (2002).

表 1: プログラムの情報と各手法の配置順序の組み合わせ数

Program	Compile Option	#Func.	#Line	Object file size [byte]	Global	Local	Basic Block	Inst.
dhry_1.c	-DHZ=100 -DTIME	6	385	7464	5.18e+05	7.20e+02	1.40e+57	1.47e+37
dhry_1.c	-DHZ=100 -DTIME -O2	6	385	7064	5.18e+05	1.00e+00	3.76e+48	4.23e+32
dhry_2.c		6	192	1936	1.00e+00	1.00e+00	1.39e+17	2.15e+10
dhry_2.c	-O2	6	192	1600	1.00e+00	1.00e+00	1.93e+14	8.06e+03
linpackc.c	-DDP -DUNROLL	12	907	15856	5.18e+05	2.69e+11	1.42e+157	6.81e+164
linpackc.c	-DDP -DUNROLL -O2	12	907	11848	5.18e+05	1.00e+00	2.74e+131	1.11e+128
whetstone.c		4	433	5984	7.20e+02	2.69e+11	5.22e+45	1.13e+43
whetstone.c	-O2	4	433	4096	7.20e+02	1.00e+00	1.94e+34	7.56e+31
ed/main.c	default option (-O)	26	1684	38704	7.22e+22	1.00e+00	2.81e+311	2.10e+77
ed/regex.c	default option (-O)	26	5171	28428	1.00e+00	1.00e+00	1.46e+337	4.60e+286
bzip2.c	default option (-O2)	43	2103	31936	3.73e+08	1.00e+00	1.98e+174	5.17e+122
gzip.c	default option (-O)	23	1744	28132	7.22e+22	1.00e+00	1.97e+134	3.82e+114
gcc.c	default option (-O2)	50	5840	77448	5.22e+45	1.00e+00	1.53e+617	7.18e+602
ldmain.c	default option (-O2)	20	1376	20512	7.20e+02	1.00e+00	3.78e+88	1.61e+88
ldlang.c	default option (-O2)	126	5525	48128	1.93e+14	1.00e+00	5.46e+345	2.39e+255