# Assignment #1 - Implement a DCE LLVM pass

R04922067 楊翔雲

1. 安裝 LLVM-3.7.1 環境
   1. 下載 LLVM 原始碼

mkdir -p ~/LLVM && cd ~/LLVM

wget -O - http://llvm.org/releases/3.7.1/llvm-3.7.1.src.tar.xz | tar Jxf -

wget -O - http://llvm.org/releases/3.7.1/cfe-3.7.1.src.tar.xz | \

tar Jxf - -C llvm-3.7.1.src/tools

* 1. 建立工作目錄

mkdir -p ~/LLVM/hw1/build

* 1. 編譯原始碼，最後會在 ~/LLVM/hw1/build 下放置 cmake\_install.cmake 檔案。

cd ~/LLVM/llvm-3.7.1.src

mkdir build && cd build

cmake ..

make -j12

sudo cmake -DCMAKE\_INSTALL\_PREFIX=`/home/morris1028/LLVM/hw1/build` -P \ cmake\_install.cmake

1. 測試流程
   1. 根據 ~/LLVM/hw1/test 下的腳本進行環境建置後，並執行 LLVM pass。

cd ~/LLVM/hw1/test

sh test.sh

sh clean.sh

* 1. 若只使用 Dead Code Elimination Pass 能移除的指令並不多。考慮先經過其他比較簡單的 Transform Pass，如 Memory to Register、Constant Propagation、Simplify the CFG，經由這些轉換後，較有機會產生出 Dead Code。
  2. 經由三次的 Transform pass 後，可以從 LLVM IR 中看到指令縮減與變換情況。也可以在 MyDCE.cpp 插入資訊，印出刪除多少個指令個數。

clang ${i}.c -DDEBUG -S -emit-llvm -o ${i}.bc

opt -mem2reg -S ${i}.bc -o ${i}-mem2reg.bc  
opt -constprop -S ${i}-mem2reg.bc -o ${i}-constprop.bc

opt -simplifycfg -S ${i}-constprop.bc -o ${i}-simplifycfg.bc

opt -load ../build/MyDCE/libMyDCE.so -myMagicDCE -S ${i}-simplifycfg.bc -o ${i}.opt.ll  
clang ${i}.opt.ll -o ${i}

1. 效能測試benchmark
   1. 測試範例 (一)  
         
      經由 clang 轉換到右方的 LLVM IR，明顯地 e = 0; 必然不會被執行，根據 Mark-and-Sweep Algorithm，由回傳值以及可能會變動外部儲存內容的指令往回推，經由 MyMagicDCE pass 後，成功移除 %e = alloca i32, align 4。  
        
      若經由 Memory to Register、Constant Propagation、Simplify the CFG 後，LLVM IR 變成如下所示，將會移除 %add = add nsw i32 5, 5

#include <stdio.h>

int main(void) {

int a = 5, e;

a = a + 5;

return 1;

e = 0;

}

define i32 @main() #0 {

entry:

%retval = alloca i32, align 4

%a = alloca i32, align 4

%e = alloca i32, align 4

store i32 0, i32\* %retval

store i32 5, i32\* %a, align 4

%0 = load i32, i32\* %a, align 4

%add = add nsw i32 %0, 5

store i32 %add, i32\* %a, align 4

ret i32 1

}

define i32 @main() #0 {

entry:

%add = add nsw i32 5, 5

ret i32 1

}

* 1. 測試範例 (二)   
       
     在這例子下，由於 store 指令具有 Side Effect，因此在毫無優化下直接執行 MyMagicDCE pass 沒辦法作任何的移除。經由 Memory to Register、Constant Propagation、Simplify the CFG 處理後，轉換後不存在任何的 Side Effect 變數，此時再藉由 MyMagicDCE pass 移除掉剩餘的殘渣。

#include <stdio.h>

int f(int \*arg) {

int ret = 0;

int a = 0, b = 0;

if (\*arg == 1)

ret = 1;

return 10;

}

int main(void) {

int v = 1;

return f(&v);

}

define i32 @f(i32\* %arg) #0 {

entry:

%0 = load i32, i32\* %arg, align 4

%cmp = icmp eq i32 %0, 1

ret i32 10

}

* 1. 沒有經過任何的 Transform Pass 下

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case \ Reduce Instruction \ Pass | MyMagicDCE | ADCE | Comment |
| Case 1 | 1 | 1 | Same |
| Case 2 | 0 | 0 | Same |

經由 Memory to Register、Constant Propagation、Simplify the CFG 後

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case \ Reduce Instruction \ Pass | MyMagicDCE | ADCE | Comment |
| Case 1 | 1 | 1 | Same |
| Case 2 | 2 | 2 | Same |

1. 參考
   1. LLVM's Analysis and Transform Passes  
      <http://llvm.org/docs/Passes.html>
   2. Carnegie Mellon University -   
      Course CS745 Optimizing Compilers Homework SPEC  
      <http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15745-s13/public/asst/asst3/asst3.pdf>
   3. 淺談編譯器最佳化技術  
      <http://www.slideshare.net/kitocheng/ss-42438227>