Analyzing Memory Accesses in x86 Executable

Compiler Construction 2004
Gogul Balakrishnan, Thomas Reps

김영철

2016. 5. 3.

Introduction

- binary-analysis tool 개발의 어려움
  - 두 addressing mode 사용하는 machine-language instructions
    explicit addressing & indirect addressing

- binary-analysis tool 이 제공해야 하는 것
  - memory location 에 대한 정보 & 조작되는 방법

- 두 가지 techniques
  - memory accesses 를 보수적으로 다룸
  - symbol-table or 디버깅 정보를 이용
    ➔ 전자 : 정확도가 낮음. 후자 : 실제로 사용하기 어려움

Introduction

- Our analysis algorithm
  - a-locs 라는 abstract data objects set 을 이용
    ➔ program data가 갖는 pointer-value, integer-value 를 추적
  - find static address(for global) & static stack-frame offsets(for local)

- Another problem
  - indirect addressing 모드 사용
  - 각각의 a-loc 이 가질 수 있는 value 의 set 을 구한다.
    ➔ flow-sensitive, context-insensitive

Introduction

- Our work
  - value-set analysis
    ➔ data object 의 value 를 추적
    ➔ abstract domain 사용
    ➔ used, killed, possibly-killed set 을 구할 수 있음
      ➔ 컴파일러의 그 것과 유사

Introduction

- example

```
int part1Value=0;
int part2Value=1;

int main() {
    int *part1,*part2;
    int a[10],*p_array0;
    int i;
    part1=&a[0];
    p_array0=part1;
    part2=&a[5];
    for(i=0;i<5;++i) {
       *part1=part1Value;
       *part2=part2Value;
       part1++;
       part2++;
    }
    return *p_array0;
}
```

```
    proc main           ;
1   sub esp, 44         ;Adjust esp for locals
2   lea eax, [esp+4]    ;part1=&a[0]
3   lea ebx, [esp+24]   ;part2=&a[5]
4   mov [esp+0], eax    ;p_array0=part1
5   mov ecx, 0          ;i=0
L1: mov edx, [4]        ;
7   mov [eax], edx      ;*part1=part1Value
8   mov edx, [8]        ;
9   mov [ebx], edx      ;*part2=part2Value
10  add eax, 4          ;part1++
11  add ebx, 4          ;part2++
12  inc ecx             ;i++
13  cmp ecx, 5          ;
14  jl L1               ;(i<5)?loop:exit
15  mov edi,[esp+0]     ;
16  mov eax, [edi]      ;set return value
17  add esp, 44         ;
18  retn                ;return *p_array0
```

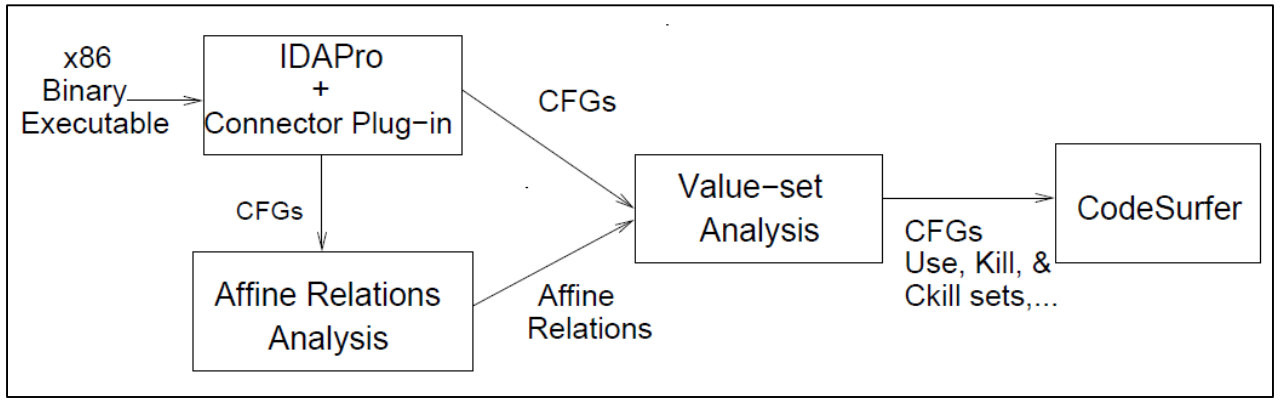
밑줄 : 'mov eax, [edi]'의 backward slicing 결과
➔ array part 를 구분 (장점)

Introduction

- value-set analysis 의 설계
  - indirect-addressing operation 이 non-aligned access 하는 것을 방지
  - 배열을 순회하는 loop 가 stack-smashing attack 하는 것을 방지
  - pointer analysis & numeric analysis 동시에 수행하도록 함
    ➔ 컴파일러가 numeric & indirect addressing 을 사용하기 때문에 중요

The Context of the Problem

- CodeSurfer/x86의 조직도



  - IDA 플러그인 Connector로 생성되는 data structure를 이용하여 VSA 구현
  - VSA 를 가지고 a-locs를 CodeSurfer의 input에 알맞은 포맷으로 생성
  - CodeSurfer는 CodeSurfer의 IR에 저장된 정보와 SDG에 접근하기 위한 GUI & API 제공

The Context of the Problem

IDA 에서 제공하는 정보

- Statically known memory address and offsets
  - a-locs 을 위해 IDA 에서 식별한 결과를 사용
- Information about procedure boundaries
  - IDA 에서 procedure 의 경계를 식별
- Calls to library functions
  - IDA FLIRT 알고리즘을 사용하여 library function 을 복원
  - malloc 에 대한 call 을 식별하기 위해 필요한 정보

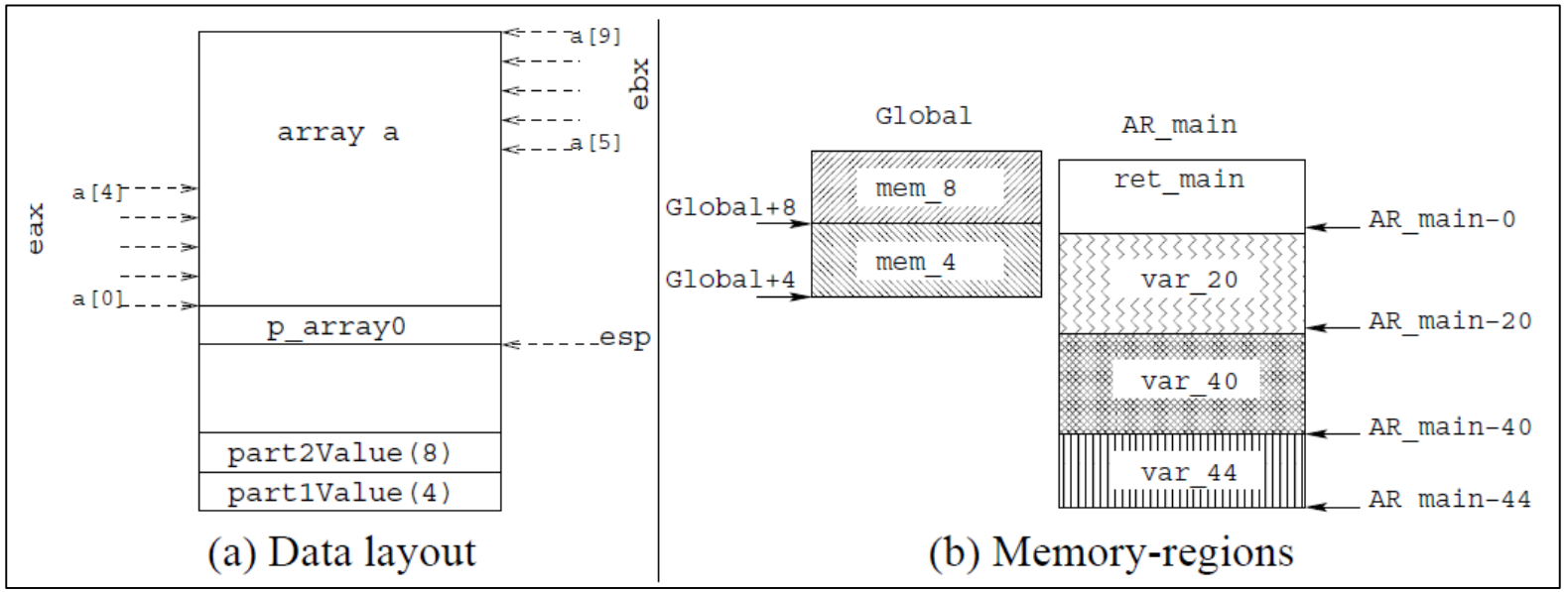
The Abstract Domain

- Abstract stores : memory-region과 a-locs 개념을 기반

- Memory-Regions
  - x-bit 머신의 메모리 주소는 x-bit 수
  - 가질 수 있는 value의 set을 구하기 위해 intervals, congruences 같은 기존의 방법을 이용
  - 문제점
    ➔ 같은 주소여도 런타임에 다른 변수를 참조할 수 있음
    ➔ 하나의 변수가 여러 개의 런타임 주소를 가질 수 있음
    ➔ 주소가 정적으로 결정되지 않을 수 있음

The Abstract Domain

- Memory-Regions
  - per procedure, per heap-allocation stmt, global region



1 procedure + 2 region

p_array0 = (AR_main, -44), part2value = (Global, 8)

The Abstract Domain

- A-Locs
  indirect addressing은 register를 포함
  register는 memory location의 value를 로드할 수 있음
  ➔ a-loc abstraction을 사용하여 추적

The Abstract Domain

- A-Locs
  - 하나의 a-loc은 하나의 변수와 같다.
  - A-loc abstraction
    ➔ 실행파일을 생성하기 전 변수들의 위치를 정한다.
    ➔ 전역변수는 direct access
    ➔ 지역변수는 indirect access (register + offset)

The Abstract Domain

- A-Locs

```
int part1Value=0;
int part2Value=1;

int main() {
    int *part1,*part2;
    int a[10],*p_array0;
    int i;
    part1=&a[0];
    p_array0=part1;
    part2=&a[5];
    for(i=0;i<5;++i) {
       *part1=part1Value;
       *part2=part2Value;
       part1++;
       part2++;
    }
    return *p_array0;
}
```

```
    proc main           ;
1   sub esp, 44         ;Adjust esp for locals
2   lea eax, [esp+4]    ;part1=&a[0]
3   lea ebx, [esp+24]   ;part2=&a[5]
4   mov [esp+0], eax    ;p_array0=part1
5   mov ecx, 0          ;i=0
L1: mov edx, [4]        ;
7   mov [eax], edx      ;*part1=part1Value
8   mov edx, [8]        ;
9   mov [ebx], edx      ;*part2=part2Value
10  add eax, 4          ;part1++
11  add ebx, 4          ;part2++
12  inc ecx             ;i++
13  cmp ecx, 5          ;
14  jl L1               ;(i<5)?loop:exit
15  mov edi,[esp+0]     ;
16  mov eax, [edi]      ;set return value
17  add esp, 44         ;
18  retn                ;return *p_array0
```

2 direct operand
3 indirect operand

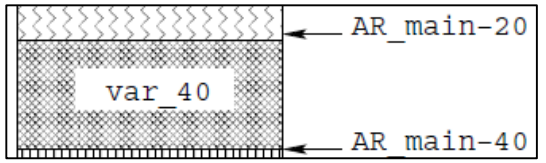
총 5개의 a-loc

The Abstract Domain

- A-Locs
  - Offset of an a-loc : *"offset(rgn, a)"*
    ex. offset(AR_main, var_20) = -20
  - Addresses of an a-loc : *"(rgn, [offset, offset+size-1])"*
    ex. var_20의 address = (AR_main, [-40, -21]) ~~*(AR_main, [-20, -1])*~~

The Abstract Domain

- Abstract Stores
  - 특정 지점에서 각각의 a-loc이 가질 수 있는 memory address의 집합
  - RIC (Reduced Interval Congruence) 사용
  - RIC는 4개의 tuple로 표현 가능
    ex. {1,3,5,9} ➔ RIC : (2Z+1)∩[0,9] ➔ 2*[0,4]+1 ➔ (2,0,4,1)
  - (a-locs -> value-set(mem->RIC)) map r : memory region의 수
    ex. eax -> value-set(⊥,4[0,4] - 40)
  - Value-set : lattice 형태



The Abstract Domain

- Abstract Stores

– $(vs_1 \sqsubseteq vs_2)$: Returns true if the value-set vs1 is a subset of vs2, false otherwise.
– $(vs_1 \sqcap vs_2)$: Returns the intersection (meet) of value-sets vs1 and vs2.
– $(vs_1 \sqcup vs_2)$: Returns the union (join) of value-sets vs1 and vs2.
– $(vs_1 \nabla vs_2)$: Returns the value-set obtained by widening vs1 with respect to vs2
– $(vs \boxplus c)$: Returns the value-set obtained by adjusting all values in vs by the constant c
– $*(vs; s)$: Returns a pair of sets $(F; P)$.
– `RemoveLowerBounds(vs)`: Returns the value-set obtained by setting the lower bound of each component RIC to ∞.
– `RemoveUpperBounds(vs)`: Similar to `RemoveLowerBounds`, but sets the upper bound of each component to ∞.

Value-Set Analysis (VSA)

- VSA
  - 실행파일의 abstract interpretation
  - abstract stores 사용
  - flow-sensitive, context-insensitive
  - pointer-analysis problem과 유사
  - 각각의 point에서 각각의 data object가 가질 수 있는 address의 집합

Value-Set Analysis (VSA)

- Intraprocedural Analysis

R1 = R2 + c	R1 $\leq c$
*(R1 + c_1) = R2 + c_2	R1 $\geq$ R2
R1 = *(R2 + c_1) + c_2	

Label on e	Transfer function for edge e
R1=R2+c	**let** $(\text{R2} \mapsto vs) \in \text{e.Before}$ $\text{e.After} := \text{e.Before} - [\text{R1} \mapsto *] \cup [\text{R1} \mapsto vs \boxplus c]$
*(R1+c_1)=R2+c_2	**let** $[\text{R1} \mapsto vs_{R1}], [\text{R2} \mapsto vs_{R2}] \in \text{e.Before}, (F, P) = *(vs_{R1} \boxplus c_1, s)$, $\text{tmp} = \text{e.Before} - \{[a \mapsto *] \mid a \in P \cup F\} \cup \{[p \mapsto \top] \mid p \in P\}$, and $Proc$ be the procedure containing the statement **if** $(\lvert F\rvert = 1$ and $\lvert P\rvert = 0$ and ($Proc$ is not recursive) and (F has no heap objects)) **then** $\text{e.After} := (\text{tmp} \cup \{[v \mapsto vs_{R2} \boxplus c_2] \mid v \in F\})$ // Strong update **else** // Weak update $\text{e.After} := (\text{tmp} \cup \{[v \mapsto (vs_{R2} \boxplus c_2) \sqcup vs_v] \mid v \in F, [v \mapsto vs_v] \in \text{e.Before}\})$

R1= *(R2+c_1)+c_2	**let** $(\text{R2} \mapsto vs_{R2}) \in \text{e.Before}$ and $(F, P) = *(vs_{R2} \boxplus c_1, s)$
	if $\lvert P\rvert = 0$ **then**
	let $vs_{rhs} = \bigsqcup\{vs_v \mid v \in F, [v \mapsto vs_v] \in \text{e.Before}\}$
	$\text{e.After} := \text{e.Before} - [\text{R1} \mapsto *] \cup [\text{R1} \mapsto (vs_{rhs} \boxplus c_2)]$
	else
	$\text{e.After} := \text{e.Before} - [\text{R1} \mapsto *] \cup [\text{R1} \mapsto \top]$

R1 $\leq c$	**let** $[\text{R1} \mapsto vs_{R1}] \in \text{e.Before}$ and $vs_c = ([-\infty, c], \top, \ldots, \top)$ $\text{e.After} := \text{e.Before} - [\text{R1} \mapsto *] \cup [\text{R1} \mapsto vs_{R1} \sqcap vs_c]$
R1 $\geq$ R2	**let** $[\text{R1} \mapsto vs_{R1}], [\text{R2} \mapsto vs_{R2}] \in \text{e.Before}$ and $vs_{lb} = \text{RemoveUpperBounds}(vs_{R2})$ $\text{e.After} := \text{e.Before} - [\text{R1} \mapsto *] \cup [\text{R1} \mapsto vs_{R1} \sqcap vs_{lb}]$

Value-Set Analysis (VSA)

- Intraprocedural Analysis

```
int part1Value=0;
int part2Value=1;

int main() {
    int *part1,*part2;
    int a[10],*p_array0;
    int i;
    part1=&a[0];
    p_array0=part1;
    part2=&a[5];
    for(i=0;i<5;++i) {
       *part1=part1Value;
       *part2=part2Value;
       part1++;
       part2++;
    }
    return *p_array0;
}
```

```
    proc main           ;
1   sub esp, 44         ;Adjust esp for locals
2   lea eax, [esp+4]    ;part1=&a[0]
3   lea ebx, [esp+24]   ;part2=&a[5]
4   mov [esp+0], eax    ;p_array0=part1
5   mov ecx, 0          ;i=0
L1: mov edx, [4]        ;
7   mov [eax], edx      ;*part1=part1Value
8   mov edx, [8]        ;
9   mov [ebx], edx      ;*part2=part2Value
10  add eax, 4          ;part1++
11  add ebx, 4          ;part2++
12  inc ecx             ;i++
13  cmp ecx, 5          ;
14  jl L1               ;(i<5)?loop:exit
15  mov edi,[esp+0]     ;
16  mov eax, [edi]      ;set return value
17  add esp, 44         ;
18  retn                ;return *p_array0
```

main의| entry
{esp->(⊥,0), mem_4->(0,⊥), mem_8->(1,⊥)}

instruction 7
{esp->(⊥,-44), mem_4->(0,⊥), mem_8->(1, ⊥),
eax->(⊥,4[0,∞]-40), ebx->(⊥,4[0,∞]-20),
var_44->(⊥,-40), ecx->([0,4],⊥)}

instruction 16
{esp->(⊥,-44), mem_4->(0,⊥), mem_8->(1, ⊥),
eax->(⊥,4[1,∞]-40), ebx->(⊥,4[1,∞]-20),
var_44->(⊥,-40), ecx->([5,5],⊥), edi->(⊥,-40)}

Value-Set Analysis (VSA)

- Interprocedural Analysis
  - procedure의 형식과 call의 행동을 식별해야한다.
  - 인자들이 스택에 쌓이기 때문에 직접적으로 사용 가능하지 않다.
  - 인자를 push하는 instruction들이 함수를 호출하기 직전에 발생할 필요가 없다.

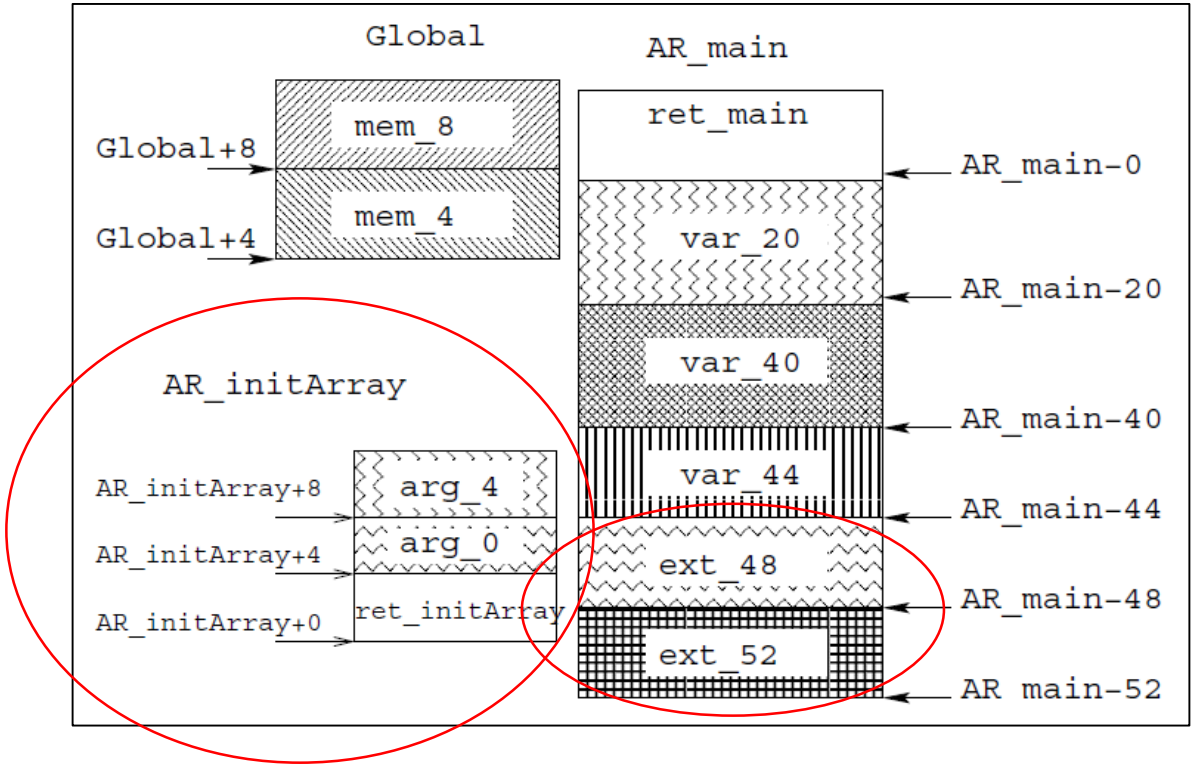
Value-Set Analysis (VSA)

- Interprocedural Analysis
  - Actual parameters and register saves
    - push/pop은 memory location을 implicit modify를 함.
    - a-locs을 식별하는 알고리즘으로는 불가
    - *"extended a-locs"* 소개
      - 최소의 sp_delta를 정해야함.

Value-Set Analysis (VSA)

- Interprocedural Analysis
  - Formal parameters
    - at entry, esp = ret
    - formal parameter
      = positive offset에 위치
    - positive offset을 갖는 a-loc

main의 extened a-locs
initArray의 formal parameters



Value-Set Analysis (VSA)

- Interprocedural Analysis
  - Handling of calls and returns
    - intraprocdural 알고리즘과 유사, +supergraph를 분석
    - supergraph : call site - call node & end-call node, 2개의 노드를 가짐.