Analyzing Memory Accesses in Object Code

Gogul Balakrishnan
University of Wisconsin - Madison
Difficulties with Object Code

Data Dependence Analysis

```c
int main()
{
    int i, j, a[10];
    j = 0;
    for(i = 0; i < 10; ++i)
    {
        a[i] = i;  // 1
    }
    return a[2];  // 2
}
```

```asm
; ebx corresponds to variable i
sub esp, 44
mov [esp+40], 0  // j = 0
xor ebx, ebx  // i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx  // a[i] = i
    inc ebx  // i++
    add ecx, 4
    cmp ebx, 10  // i < 10?
    jl short loc_9;
    mov eax, [esp+8]  // return a[2]
    add esp, 44
    retn
```

Affects?
Yes!

Affects?
Yes!
Difficulties with Object Code

• Access/update of memory in object code
  - By specifying the addresses
    • Directly/indirectly

• Tracking data for static analysis
  - Should track data in addresses
  - Difficult in object code
    • Large number of addresses to track
    • Sometimes addresses are symbolic
      - Local variable is an offset in the stack frame
      - Start of the stack frame is dynamic
Difficulties with Object Code

- Variables in HLL programs
  - A manageable domain for static analysis
  - A set of similar runtime locations
  - Ex. Local variable
- Need a similar mechanism in object code
- The entity that represents a set of similar runtime locations
  - Abstract Location (ALOC)
Existing Infrastructure

Object Code → IDAPro

Disassembly & associated information

Converter → IDAPro SDK

Pre-IR

def, use, cond-kill sets etc.,

C Programs → Codesurfer

Codesurfer

Virus Scanner

Buffer Overrun Detector

Codesurfer datastructures (SDGs, CFGs etc.)
Clients of the Analysis

- **Mihai’s virus scanner**
  - Memory accesses are treated conservatively
  - With ALOCs data in memory can be included in the analysis
    - Detection of obfuscations is improved
    - Removal of irrelevant statements is improved
  - Virus specification in high-level terms (using ALOCs)
Clients of the Analysis

• Jon’s intrusion detection system
  - Captures static values of parameters to improve the model
  - Better slicing can help in this regard

• Vinod’s buffer overrun tool
  - Can be adapted to object code
Abstract Interpretation

• At each program point
  - Determine what addresses are accessed
  - Keep track of the set of addresses a register holds

• Use the *Abstract Interpretation Framework* of P. Cousot and R. Cousot
Abstract Interpretation

• Static analysis technique
  - To identify dynamic properties of the variables in a program at compile time
  - The results are approximate, but safe

• Abstract interpretation
  - Runtime values approximated by abstract values
  - Program execution simulated on abstract values
  - Iterated until fixed point is reached
  - Ex: Constant propagation
Abstract Interpretation

• For our case,
  - Runtime value is a set of addresses in a register
  - Cardinality of the set is very large

• Represent the sets by safe approximations
  - Safe ⇒ Can have more addresses but should not miss any address
Abstract Interpretation

- **Abstract Domain**
  - A practical representation of runtime values and operators to manipulate the representation
  - Operators should reflect the semantics of the programming language

- **Terminology**
  - Runtime value: *Concrete value*
  - Approximate representation: *Abstract value*
  - Concrete value->Abstract value: *Abstraction*
  - Abstract value->Concrete value: *Concretization*
Abstract Domain - Examples

• **Intervals** - \([a,b]\)
  - Represents any set \(\subseteq \{x | a \leq x \leq b\}\)
  - Abstraction\((S)\): \([\min(S), \max(S)]\)
  - \(\{3, 6, 9, 15\}\) - represented as \([3, 15]\)
  - \([3, 15] \supseteq \{3, 6, 9, 15\}\)
  - Cannot represent arbitrary strides
Abstract Domain - Examples

- **Congruences** - $a \mathbb{Z} + b$
  - Represents any set $\subseteq \{a \times x + b | x \in \mathbb{Z}\}$
  - Abstraction($S$): $\text{gcd}(|x| x \in S) \mathbb{Z} + \min(|x| x \in S)$
  - $\{3, 6, 9, 15\}$ - represented as $3\mathbb{Z} + 3$
  - $3\mathbb{Z} + 3 = \{..., -3, 0, 3, ...\} \supseteq \{3, 6, 9, 15\}$
  - Captures stride information to an extent
  - But loses bounds information
Relational vs Non-Relational Domains

• Non relational domains
  - Cannot capture relationship among registers
  - Ex: Intervals
    - $R_i \in [C_1, C_2]$

• Relational domains
  - Can capture relationship among registers
Examples of Relational Domains

<table>
<thead>
<tr>
<th>Intervals (Non-Relational)</th>
<th>Difference Constraints</th>
<th>Octagonal Constraints</th>
<th>Polyhedra</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1 \leq c_1, R_1 \geq c_2$, $R_2 \leq c_3, R_2 \geq c_4$</td>
<td>$R_1 - R_2 \leq C$</td>
<td>$\pm R_1 \pm R_2 \leq C$</td>
<td>$a_1 R_1 + a_2 R_2 + \ldots + a_n R_n \leq C$</td>
</tr>
</tbody>
</table>

Faster  More precise
Weakly Relational Domain (WRDomain)

- Proposed by A. Miné (SAS 2002)
- Capable of expressing relations of the form
  - \( R_1 - R_2 \in C \)
  - \( C \in \text{Basis} \) (another abstract domain)
- Basis
  - Parameterizes WRDomain
  - Different basis domains \( \Rightarrow \)
    - Different WRDomains
    - Different precision and efficiency
  - Eg., Interval, Congruence
- Between polyhedra and difference constraints
  - in terms of accuracy and efficiency
Weakly Relational Domain - Example

- Interval-based WRDomain
  - The basis is the interval domain
  - Can express relations like $R_1 - R_2 \in [c_1, c_2]$

```assembly
mov edx, 10
mov eax, ecx
L1: dec eax
dec edx
jnz L1
```

```
edx = [1,10]
cecx - eax = [0,9]
```
Weakly Relational Domain - Example

- Interval-based WRDomain
  - The basis is the interval domain
  - Can express relations like $R_1 - R_2 \in [c_1, c_2]$
Advantages of WRDomain

• Reasonable time complexity and accuracy
  - $O(n^3)$, $n$ is the number of variables

• Clients are diverse
  - Have different requirements
    • Precision
    • Efficiency
  - \[\therefore\] need tunable precision
    • Have several implementations
    • Or, implement *one* flexible *framework* \[\leftarrow\]
      - WRDomain allows flexibility through the choice of Basis
A Sneak Peek at the Implementation

- Flexible implementation using templates
- template <class Basis>
  class WRDomain<Basis>
- WRDomain implements the operators
- How to implement WRDomain with different bases?
  - Implement Basis class
  - Instantiate WRDomain class with that Basis class
Abstract Interpretation - Using Weakly Relational Domains

• Assumptions
  - Program accesses only data on stack frame
  - Treat memory accesses conservatively for the time-being
    • mov ecx, [esp+10]
    • After this statement ecx becomes unknown
Abstract Interpretation - Example

```
; ebx corresponds to variable i
sub esp, 44
mov [esp+40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i] = i
    inc ebx ; i++
    add ecx, 4
    cmp ebx, 10 ; i<10?
    jl short loc_9 ;
    mov eax, [esp+8] ; return a[2]
    add esp, 44
    retn
```
**Interval-based WRDomain - Example**

ecx - AR ∈ [-44,-8] \[⇒ ecx = AR - \{44,43,...,8}\]

All addresses - part of array a

esp - AR ∈ [-44,-44] \[⇒ (esp + 8) = AR - 36\]

Address refers to the middle of array a

```
; ebx corresponds to variable i
sub esp, 44
mov [esp+40],0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
  mov [ecx], ebx ; a[i]=i
  inc ebx ; i++
  add ecx, 4
  cmp ebx, 10 ; i<10?
  jl short loc_9 ;
  mov eax, [esp+8] ; return a[2]
  add esp, 44
  ret
```
Problems with Interval-based WRDomain

- **Overlap in access?**
  - `mov [ecx],eax`, modified the words in memory addresses AR + [-44,-8]
  - Does not answer if words overlapped

---

Jan 28, 2003
Gogul Balakrishnan - bgogul@cs.wisc.edu
Problems with Interval-based WRDomain

- **Struct objects**
  - Suppose `a` is struct `{int x,y}` `a[10];`
  - Consider the memory operand `a[i].x`
  - Interval-based WRDomain ⇒ The whole struct object is accessed

![Diagram showing the access to a struct object using interval-based WRDomain]

*Actually!*
Problems with Interval-based WRDomain

• Basic problem in two cases
  - No stride information
• Congruence can capture stride
• Use congruence-based WRDomain
**Congruence-based WRDomain - Example**

\[ \text{ecx} - \text{AR} \in (4\mathbb{Z}+0) \quad \Rightarrow \text{ecx} = \text{AR} - \{-4, 0, 4, \ldots\} \]

\[ \text{esp} - \text{AR} \in \mathbb{Z}-44 \quad \Rightarrow \text{(esp + 8)} = \text{AR} - 36 \]

Successfully determined stride. But lost bounds!!

Jan 28, 2003  
Gogul Balakrishnan - bgogul@cs.wisc.edu
Product Domain

• Product Domain
  - Combination of two different domains
  - A standard technique in abstract interpretation
  - Produces better results than the two domains

• A product domain can be a Basis of a WRDomain

• Product domain of interval and congruence domains
  - Represent values in the form $a\star [b,c]+d$
Product domain-based WRDomain - Example

ecx - AR ∈ (4*[0,9]-44)  
⇒ ecx = AR - {44, 40,…,8}

esp - AR ∈ ∞*[0,0]-44  
⇒ (esp + 8) = AR - 36

; ebx corresponds to variable i
sub esp, 44
mov [esp+40],0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
mov [ecx], ebx ; a[i]=i
inc ebx ; i++
add ecx, 4
cmp ebx, 10 ; i<10?
jl short loc_9 ;
mov eax, [esp+8] ; return a[2]
add esp, 44
retn
Tracking Data in Memory

- Registers can get data from memory
- For better precision, capture data-flow through memory
- But, large number of addresses & symbolic addresses
  - So, track only statically known addresses
    - Like variables in HLL program
    - But not exact
Tracking Data in Memory - Example

• Identify static addresses

; ebx corresponds to variable i
sub esp, 44
mov [esp+40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i]=i
    inc ebx ; i++
    add ecx, 4
    cmp ebx, 10 ; i<10?
    jl short loc_9 ;
mov eax, [esp+8] ;return a[2]
add esp, 44
retn
Tracking Data in Memory - Example

- Identify static addresses
- Name them consistently
- Include in the WRDomain

```assembly
; ebx corresponds to variable i
sub esp, 44
mov [esp+var_40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i]=i
    inc ebx ; i++
    add ecx, 4
    cmp ebx, 10 ; i<10?
    jl short loc_9 ;
    mov eax, [esp+var_8] ;return a[2]
    add esp, 44
    ret
```
Tracking Data in Memory - Example

; ebx corresponds to variable i
sub esp, 44
mov [esp+var_40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i]=i
    inc ebx ; i++
    add ecx, 4
    cmp ebx, 10 ; i<10?
    jl short loc_9 ;
    mov eax, [esp+var_8] ; return a[2]
    add esp, 44
    retn
Tracking Data in Memory - Example

• Identify static addresses
• Name them consistently
• Include in the WRDomain
• For a memory operand
  • Let R - esp = C
  • Find memory variables in range esp + C
• Update info about them

Ex:
  • mov [ecx], ebx
  • ecx-esp \epsilon 4*[0,9]-44
  • Update var_8 - ebx to 0

; ebx corresponds to variable i
sub esp, 44
mov [esp+var_40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
  mov [ecx], ebx ; a[i]=i
  inc ebx ; i++
  add ecx, 4
  cmp ebx, 10 ; i<10?
  jl short loc_9 ;
  mov eax, [esp+var_8] ;return a[2]
  add esp, 44
  retn
Discovering ALOCs

• All memory locations accessed through an operand
  - Should have the same type
  - If contiguous, they probably belong to the same object
  - Longest such contiguous sequence - ALOC

• Identify ALOCs from the results of abstract interpretation

• Now annotating use, def and c.kill sets is easy
Discovering ALOCs - Example

\[ \text{esp} - \text{AR} \in \mathbb{R} \times [0,0] - 44 \]
\[ \Rightarrow (\text{esp} + 40) = \text{AR} - 4 \]
\[ \Rightarrow \text{AR} + [-4, -1] \text{ is an ALOC} \]

\[ \text{ecx} - \text{AR} \in (4 \times [0,9] - 44) \]
\[ \Rightarrow \text{ecx} = \text{AR} - \{44, 40, \ldots, 8\} \]
\[ \Rightarrow \text{AR} + [-44, -5] \text{ is an ALOC} \]

; ebx corresponds to variable i
sub esp, 44
mov [esp+40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
  mov [ecx], ebx ; a[i]=i
  inc ebx ; i++
  add ecx, 4
  cmp ebx, 10 ; i<10?
  jl short loc_9 ;
mov eax, [esp+8] ;return a[2]
add esp, 44
retn
Discovering ALOCs - Example

```
int main()
{
    int i, j, a[10];
    j=0;
    for(i=0;i<10;++i)
    {
        a[i]=i;
    }
    return a[2];
}
```
Discovering ALOCs - Example

int main(){
    int i, j, a[10];
    j=0;
    for(i=0;i<10;++i){
        a[i]=i;
    }
    return a[2];
}
Work Done So Far

• Interval and Congruence-based WRDomains
• Only memory accesses in local stack frame

Work in Progress

• Product domain of interval and congruence
• Considering memory access to global data area and stack frame of other procedures
Future Work

• Inter-procedural Context-Sensitive Analysis
  - Based on Sharir and Pnueli summary functions

• Discovering Type Information
  - Using aggregate structure identification
  - G. Ramalingam et al (POPL 99)
  - Ignore declarative information
  - Identify fields from the access patterns
Future work - Discovering Type Information

AR[−44:−5]

40

; ebx corresponds to variable i
sub esp, 44
mov [esp+40],0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i]=i
    inc ebx ; i++
    add ecx, 4
    cmp ebx, 10 ; i<10?
    jl short loc_9 ;
    mov eax, [esp+8] ;return a[2]
    add esp, 44
    retn
Future work - Discovering Type Information

; ebx corresponds to variable i
sub esp, 44
mov [esp+40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i]=i
    inc ebx ; i++
    add ecx, 4
    cmp ebx, 10 ; i<10?
    jl short loc_9 ;
    mov eax, [esp+8] ;return a[2]
    add esp, 44
    retn
Future work - Discovering Type Information

; ebx corresponds to variable i
sub esp, 44
mov [esp+40], 0 ; j = 0
xor ebx, ebx ; i = 0
lea ecx, [esp]
loc_9:
    mov [ecx], ebx ; a[i]=i
    inc ebx ; i++
    add ecx, 4
cmp ebx, 10 ; i<10?
jl short loc_9 ;
    mov eax, [esp+8] ; return a[2]
    add esp, 44
    ret
Preparing Object Code for Static Analysis

Gogul Balakrishnan
University of Wisconsin - Madison