Analyzing Memory Accesses in Object Code

Gogul Balakrishnan

University of Wisconsin - Madison

Difficulties with Object Code

Data Dependence Analysis

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

Affects?
Yes!

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

Affects?
???##!!!

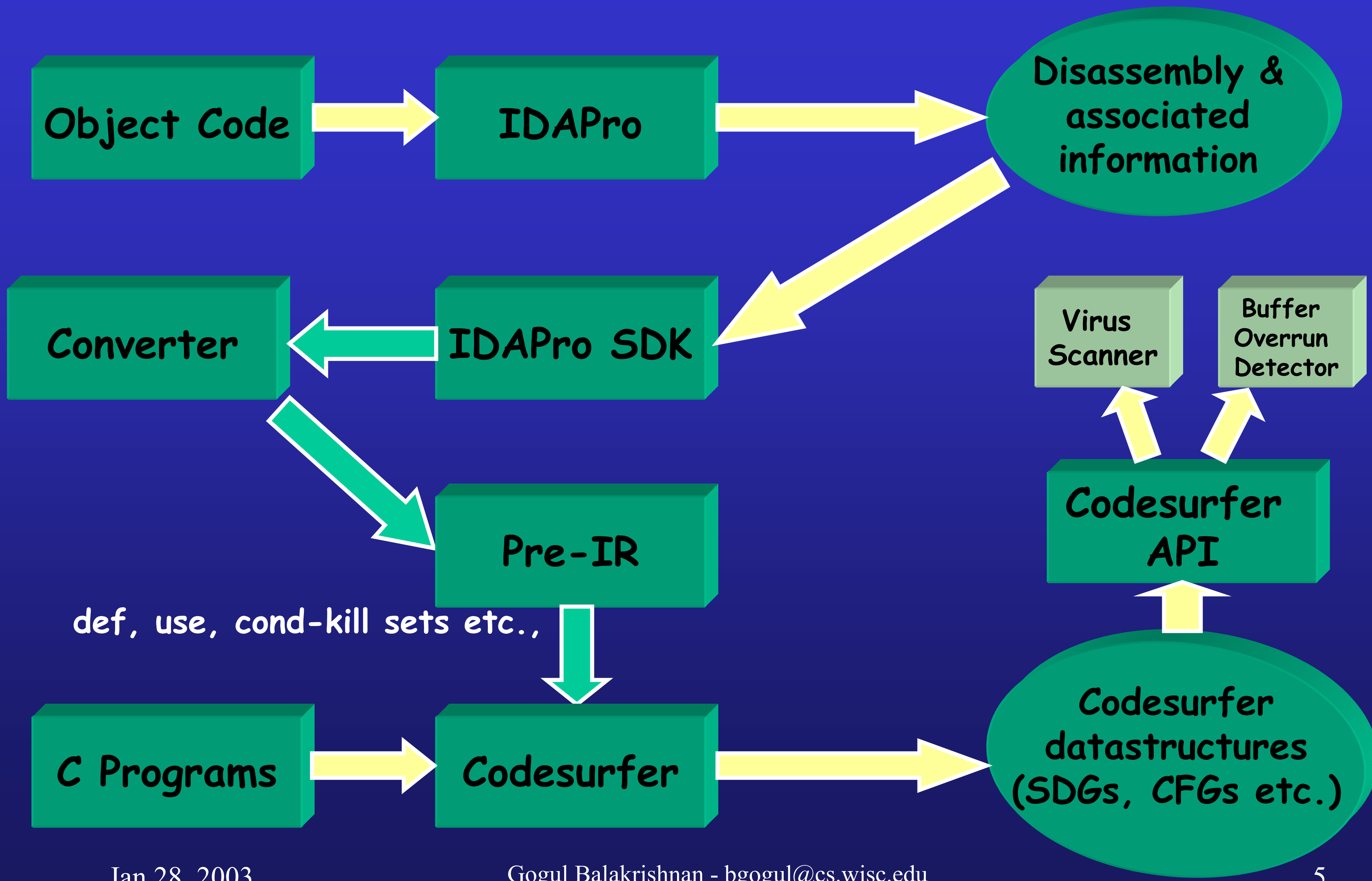
Difficulties with Object Code

- 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



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 <u>safe</u> approximations
  - Safe $\Rightarrow$ Can have more addresses but should <u>not</u> 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 $\subseteq \{x \mid a \leq x \leq b\}$
  - 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_*Z+b
  - Represents any set $\subseteq \{a_*x+b \mid x \in Z\}$
  - Abstraction(S): $gcd(\{x \mid x \in S\})Z+min(\{x \mid x \in S\})$
  - $\{3,6,9,15\}$ – represented as $3Z+3$
  - $3Z+3=\{\ldots,-3,0,3,\ldots\} \supseteq \{3,6,9,15\}$
  - Captures stride information to an extent
  - But loses bounds information

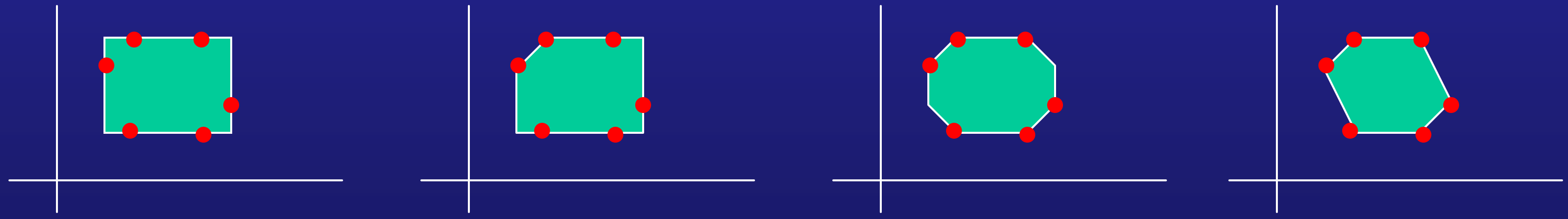
Relational vs Non-Relational Domains

- Non relational domains
  - Cannot capture relationship among registers
  - Ex: Intervals
    - Ri ∈ $[C_1, C_2]$
- Relational domains
  - Can capture relationship among registers

Gogul Balakrishnan - bgogul@cs.wisc.edu

Examples of Relational Domains

Intervals (Non-Relational)	Difference Constraints	Octagonal Constraints	Polyhedra
$R_1 \leq c_1$, $R_1 \geq c_2$ $R_2 \leq c_3$, $R_2 \geq c_4$	$R1 - R2 \leq C$	$\pm R1 \pm R2 \leq C$	$a_1 R_1 + a_2 R_2 + \ldots + a_n R_n \leq C$



Faster ← → More precise

Weakly Relational Domain (WRDomain)

- Proposed by A. Miné (SAS 2002)
- Capable of expressing relations of the form
  - $R_1 - R_2 \in C$
  - $C \in$ Basis(another abstract domain)
  - Basis
    - Parameterizes WRDomain
    - Different basis domains ⇒
      - 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 $R_1 - R_2 \in [c_1, c_2]$

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

```
edx = [1,10]
ecx - eax = [0,9]
```

Weakly Relational Domain - Example

- Interval-based WRDomain
  - The basis is the interval domain
  - Can express relations like R1–R2 ∈ [c1,c2]

	Intervals	WRDomain

mov edx, 10
mov eax, ecx
L1:dec eax
dec edx
jnz L1
	edx ∈ [1,10]
	eax ∈ [-∞,+∞]
	ecx ∈ [-∞,+∞]
		edx – 0 ∈ [1,10]
		ecx - eax ∈ [0,9]

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
  - ∴ need tunable precision
    - Have several implementations
    - Or, implement ***one*** flexible ***framework*** <---
      - WRDomain allows flexibility through the choice of Basis

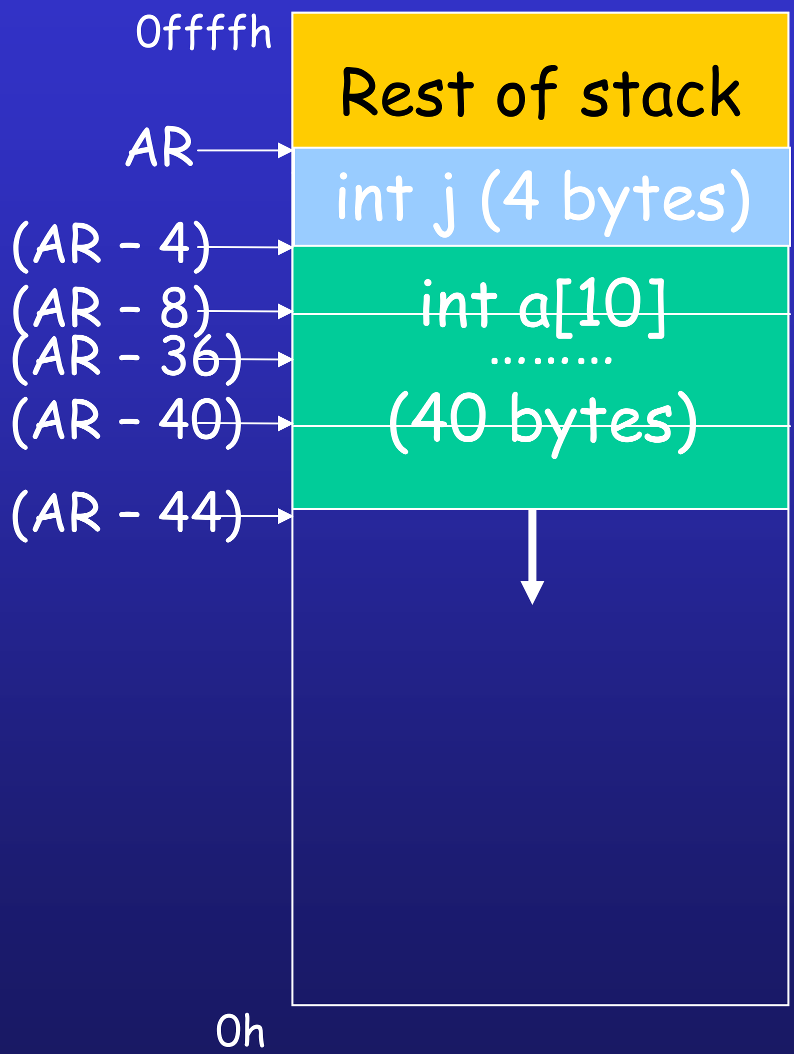
A Sneak Peek at the Implementation

- 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

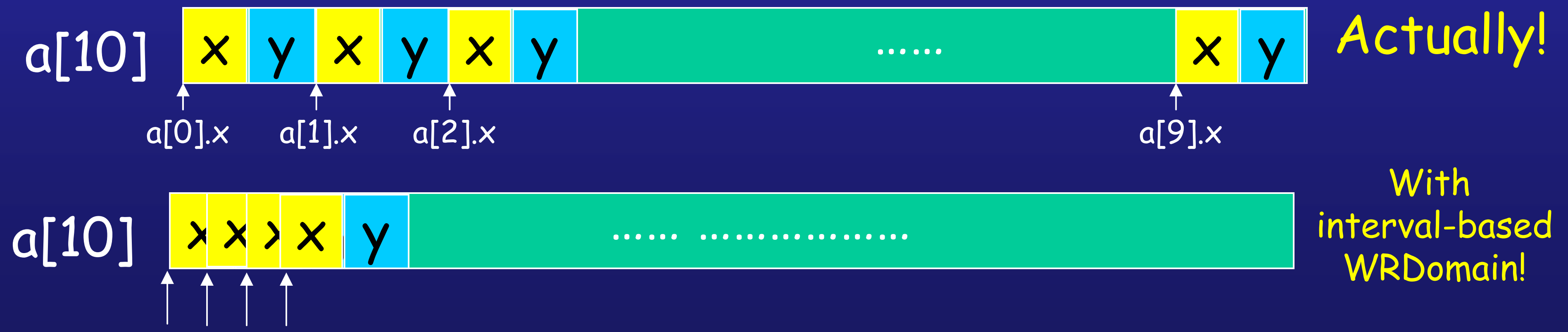
```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0   ; j = 0
xor     ebx, ebx     ; i = 0
lea     ecx, [esp]
loc_9:
   mov     [ecx], ebx ; a[i]=i
   inc     ebx         ; i++
   add     ecx, 4
   cmp     ebx, 10      ; i<10?
   jl      short loc_9 ;
mov     eax, [esp+8] ;return a[2]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

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

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



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

ecx – AR ∈ (4*Z+0)
⇒ecx = AR – {…,-4,0,4,…}

esp – AR ∈ ∞*Z-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
```

Successfully determined stride.
But lost bounds!!

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*[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

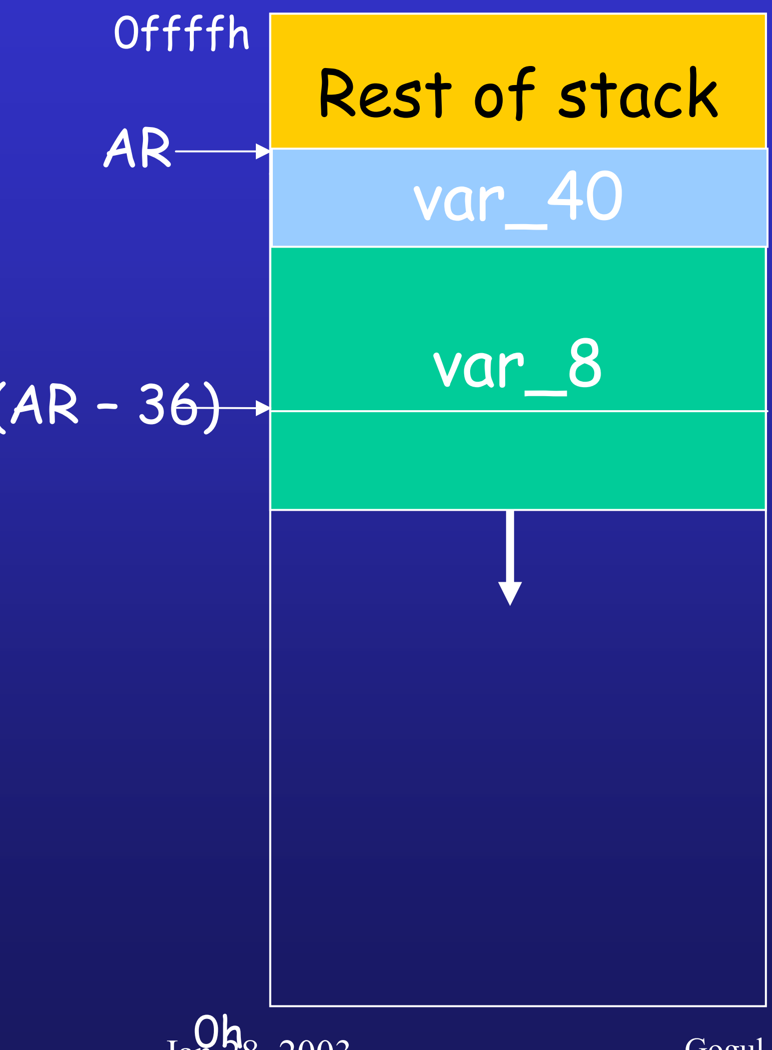
```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0    ; j = 0
xor     ebx, ebx      ; i = 0
lea     ecx, [esp]
loc_9:
   mov     [ecx], ebx ; a[i]=i
   inc     ebx          ; i++
   add     ecx, 4
   cmp     ebx, 10       ; i<10?
   jl      short loc_9 ;
mov     eax, [esp+8] ;return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

Tracking Data in Memory - Example

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

```
; 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




```
; 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 Є 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

esp – AR ∈ ∞*[0,0]-44
⇒ (esp + 40) = AR - 4

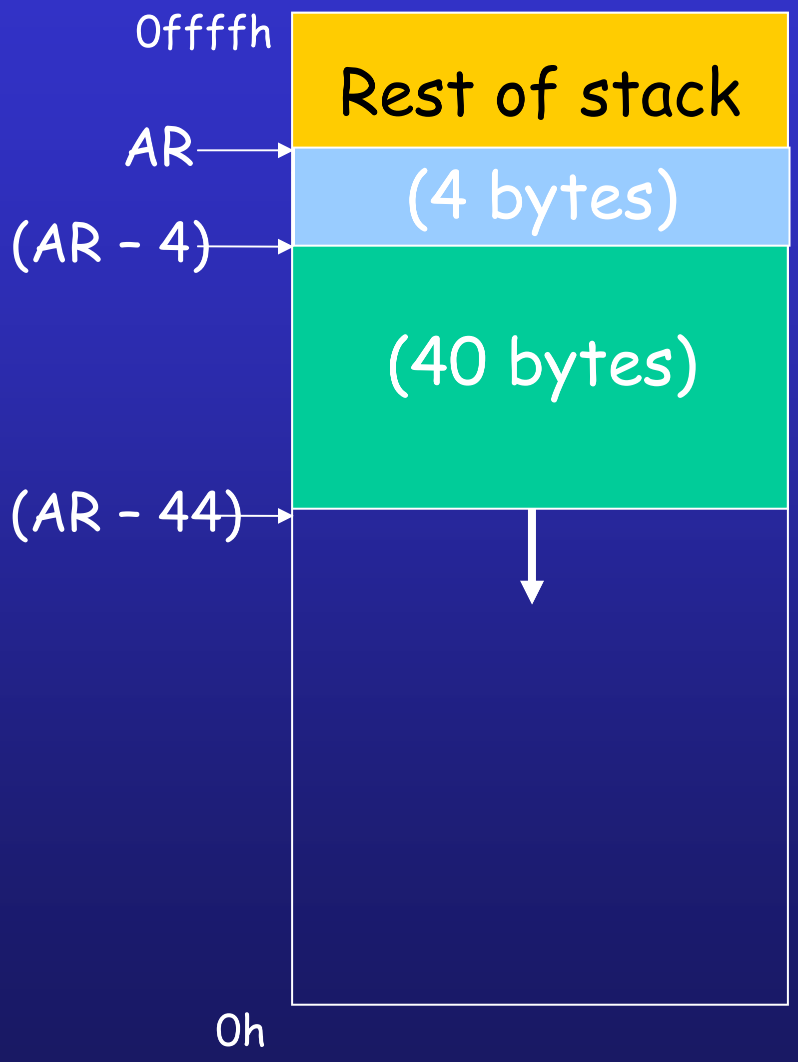
⇒AR+[-4,-1] is an ALOC

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

⇒AR+[-44,-5] is an ALOC

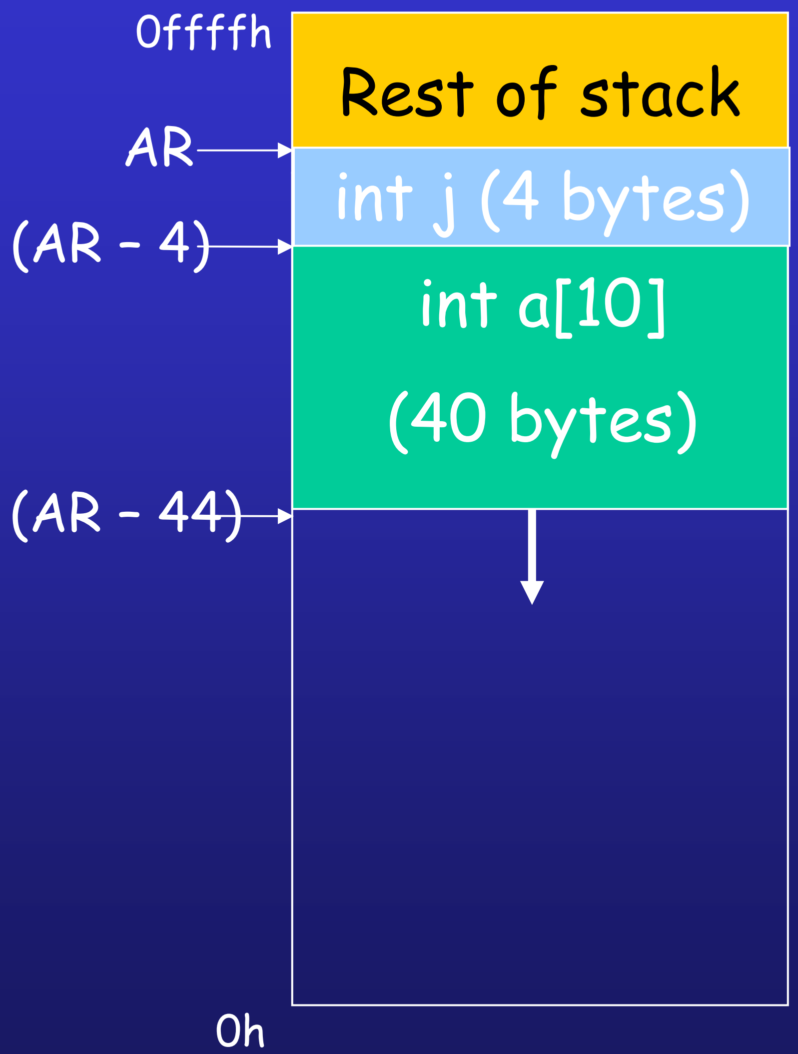
```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0   ; j = 0
xor     ebx, ebx     ; i = 0
lea     ecx, [esp]
loc_9:
   mov     [ecx], ebx ; a[i]=i
   inc     ebx          ; i++
   add     ecx, 4
   cmp     ebx, 10      ; i<10?
   jl      short loc_9 ;
mov     eax, [esp+8] ;return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

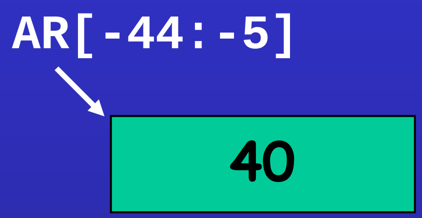
Work in Progress

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

Future Work

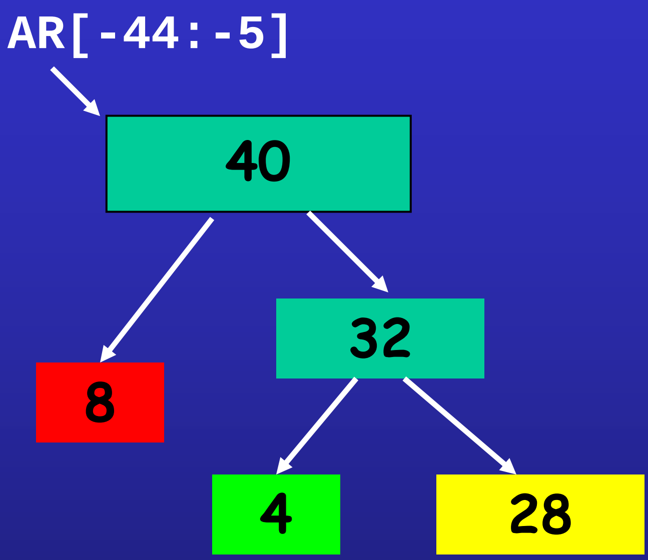
- Inter-procedural Context-Sensitive Analysis
  - Based on Sharir and Pnueli summary functions
- Discovering Type Information
  - Using aggregate structure identification
  - G. Ramalingam et al (POPL 99)
  - Ignore declarative information
  - Identify fields from the access patterns

Future work – Discovering Type Information



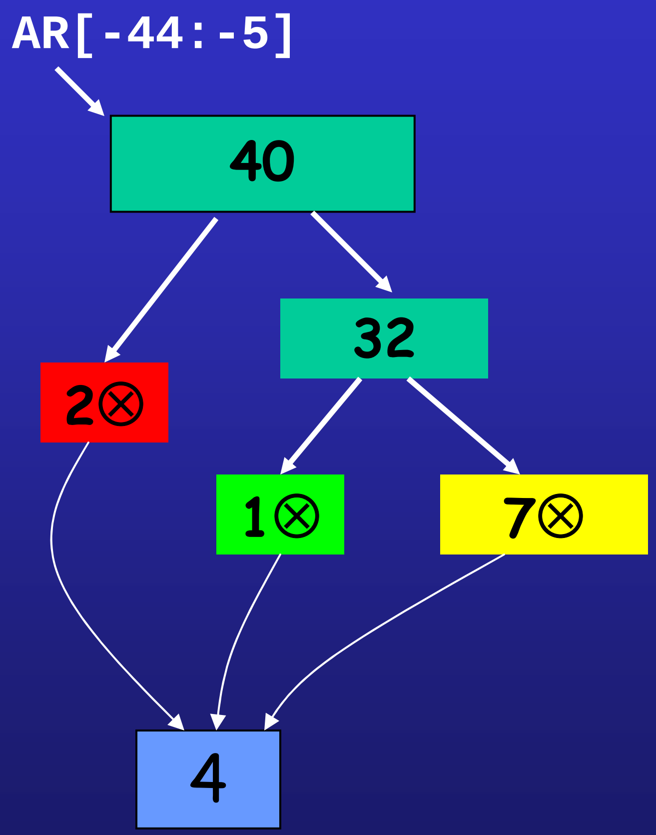

```
; ebx corresponds to variable i
sub     esp, 44
mov     [esp+40],0   ; j = 0
xor     ebx, ebx     ; i = 0
lea     ecx, [esp]
loc_9:
   mov     [ecx], ebx ; a[i]=i
   inc     ebx          ; i++
   add     ecx, 4
   cmp     ebx, 10       ; i<10?
   jl      short loc_9 ;
mov     eax, [esp+8] ;return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

Preparing Object Code for Static Analysis

Gogul Balakrishnan
University of Wisconsin - Madison